

人工智能及其应用丛书

人工智能导论

林尧瑞 马少平 编著

清华大学出版社

432013

人工智能导论

林尧瑞 马少平 编著



00432015



清华大学出版社

内 容 简 要

本书主要讲述人工智能问题求解方法的一般性原理和基本思想，并简要介绍人工智能语言，知识表示方法以及几个应用领域中所涉及的人工智能问题。

全书共十章，前五章是人工智能基本原理和方法的论述，后五章是人工智能技术应用中涉及的几个主要问题，这些内容都是了解人工智能技术最基本的入门知识。

本书可作为计算机专业本科生和研究生学习人工智能导论课程的教材和参考书，也可作为其他专业研究生了解人工智能技术的参考书。本书还可作为其他对人工智能技术有兴趣的科技、工程技术人员学习的入门参考书。

(京)新登字158号

人 工 智 能 导 论

林尧瑞 马少平 编著



清华大学出版社出版

北京 清华园

北京昌平县振南排版厂排版

北京人民文学印刷厂印刷

新华书店总店科技发行所发行



开本：850×1168 1/32 印张：11 5/8 字数：302 千字

1989年5月第1版 1990年7月第9次印刷

印数：34001~38000

ISBN 7-302-00408-0/TP·135

定价：12.80 元

人工智能及其应用丛书编委会

主任：常 迥

副主任：张 镊

委员： (按姓氏笔划)

边肇祺 朱雪龙 吴中权 李衍达

李幼哲 林尧瑞 周远清 徐光谱

人工智能及其应用丛书书目

1. 人工智能导论
2. 人工智能原理
3. 人工智能基础
4. 专家系统原理与实践
5. 知识工程——知识库系统
6. 人工智能程序设计
7. 智能机器人
8. 计算机视觉及其应用
9. 计算机语音处理与识别
10. 智能控制系统
11. 图形的识别与理解
12. 自然语言理解

出版说明

人工智能自 1956 年问世以来的三十年间已经取得了一些进展，并正在引起越来越多人的重视。它的基础研究难度大，如知识表示、推理方法、机器学习等问题虽取得了一些成果，但还远未形成完整的理论和体系。在应用研究方面，近十多年来成绩显著，像大家熟知的专家系统，已广泛应用于各种性质不同的领域，从工业、农业到医学，从商业、教育到军事，都取得了不少实际成效，有的还形成商品投入市场。这些进展正吸引着各类专家加入人工智能的研究队伍，他们之中有计算机学家、心理学家、语言学家、数学家、哲学家和各部门的工程技术人员。近几年来还有为数不少的企业家也加入这支庞大的队伍。世界各地已成立了几十、上百个销售人工智能产品的企业和商号，组成了一支号称“人工智能商业”的队伍。这股研究和开发人工智能技术的热潮目前也正在我国兴起。

人工智能为什么具有这么大的吸引力？与其说由于它的已有成就，不如说是由于它的潜在能力。专家们已经看到，人工智能将使计算机能够解决那些至今人们还不知道如何解决的问题，从而大大地扩充其用途。它将带来计算机硬件和软件的革命。人工智能正向各个领域渗透，带来这些领域的更新换代。智能计算机管理，智能计算机辅助设计，智能机器人等新的研究领域，由于人工智能的发展而不断出现。人工智能的发展还有助于我们进一步理解人类智能的机制。这一切都将促进和加快社会经济的发展，因此受到各国的普遍重视。

为了满足广大读者的需要，我们组织编写了《人工智能及其应用》这部丛书并准备通过这部丛书向大家介绍人工智能的基本原理及其主要应用技术。包括人工智能原理与基础，知识工程与

专家系统，计算机视觉，计算机语言学，计算机语音处理与识别，智能机器人学以及智能控制等内容。全书将分十多册在两、三年内陆续出版。

清华大学自1979年就开始人工智能的教学和科研工作。十年来，共开设了十多门与人工智能有关的本科生与研究生的课程，进行了几十项人工智能的科研项目。这部丛书的内容基本上来自上述工作，丛书的作者都是上述教学、科研工作的参加者。多数书是集体合作写成的，仅署名的作者就有几十位。有的内容取自本科生或研究生的论文，实验研究的成果，所以确切地说，这部丛书是集体劳动的成果。

由于人工智能是一门正在发展中的学科，尚未形成自己的完整体系，所以收集在这部丛书里的内容，不可能全是系统和成熟的，这些将留给读者自己去判断。

这部丛书可作为计算机、自动化、无线电等有关专业的本科生或研究生的教材，也可作为一般工程技术人员和科学工作者的参考书。

欢迎广大读者对本书提出批评与改进意见。

前　　言

人工智能是近 30 年来发展起来的一门综合性学科，从目前发展的趋势来看，它不仅具有实用价值，而且愈来愈引起其它学科专业人员的兴趣和重视。编写本书目的之一是通过介绍人工智能基本思想和方法，为计算机科学和技术人员以及其他学科领域中对人工智能感兴趣的科技工作者提供最基本的人工智能技术和有关问题的入门知识。

本书前五章主要讲述人工智能问题求解的基本方法，重点是阐明这些方法的一般性原理和主要思想，没有涉及人工智能具体的应用课题。后五章简要介绍人工智能语言、知识表示方法以及几个应用领域中涉及到的人工智能问题。这些内容都是学习人工智能技术最基本的知识，因此本书较适合于作为计算机专业的本科生学习人工智能导论课程的教材。此外，本书也可作为其它对人工智能技术有兴趣的科技工作者学习人工智能的入门参考书。

本书的内容可在一学期内讲授完，每章的末尾都附有与讲课内容紧密结合的习题，供学生在学习过程中深入消化基本概念时练习，有些题目也可作为课堂讨论的内容。

本书在写作过程中得到教研组许多同志的帮助和鼓励，在此表示感谢。对书中的缺点和错误欢迎读者批评指正。

清华大学计算机系

林尧瑞 马少平

1987.12

目 录

绪论	1
0.1 人工智能的研究目标	1
0.2 人工智能发展简史	3
0.3 人工智能研究的课题	7
第一章 产生式系统	14
1.1 产生式系统的组成部分	14
1.2 产生式系统的基本过程	23
1.3 产生式系统的控制策略	23
1.4 问题的表示	32
1.5 产生式系统的类型	35
1.6 小结	47
习题.....	48
第二章 产生式系统的搜索策略	50
2.1 回溯策略 (Backtracking Strategies)	51
2.2 图搜索策略	57
2.3 无信息图搜索过程	60
2.4 启发式图搜索过程	61
2.5 搜索算法讨论	89
2.6 小结	97
习题.....	98
第三章 可分解产生式系统的搜索策略	101
3.1 与或图的搜索	101
3.2 与或图的启发式搜索算法AO*	105
3.3 博弈树的搜索	110
3.4 小结	123

习题	124
第四章 人工智能中的谓词演算及应用	126
4.1 一阶谓词演算的基本体系	126
4.2 归结（消解Resolution）	129
4.3 归结反演系统（Refutation）	138
4.4 基于归结法的问答系统	144
4.5 基于归结的自动程序综合	153
4.6 基于归结的问题求解方法	157
4.7 基于规则的正向演绎系统	162
4.8 基于规则的逆向演绎系统	172
4.9 基于规则的演绎系统的几个问题	180
4.10 小结	181
习题	182
第五章 人工智能系统规划方法	187
5.1 规划（Planning）	187
5.2 机器人问题求解	189
5.3 规划的表示问题	195
5.4 使用目标堆栈的简单规划方法	198
5.5 用目标集的非线性规划方法	208
5.6 分层规划方法	214
5.7 小结	223
习题	224
第六章 人工智能语言	227
6.1 LISP	228
6.2 PLANNER	255
6.3 PROLOG	258
6.4 专家系统工具	263
6.5 小结	269

习题	270
第七章 知识表示	271
7.1 单元表示	272
7.2 语义网络	279
7.3 概念从属	283
7.4 框架	290
7.5 脚本	295
7.6 过程表示	299
7.7 小结	303
习题	303
第八章 自然语言理解	305
8.1 引言	305
8.2 简单句理解	311
8.3 复合句理解	316
8.4 语言生成	320
8.5 机器翻译	321
8.6 小结	322
习题	323
第九章 感知	325
9.1 感知问题概述	325
9.2 求解感知问题所使用的技术	327
9.3 约束满足法	329
9.4 小结	339
习题	339
第十章 学习	340
10.1 概述	340
10.2 机器学习的分类	342
10.3 机械（或死记）学习（Rote Learning）	345

10.4 指点或教授学习 (Learning by being told)	347
10.5 类比学习 (Learning by Analogy)	348
10.6 概念学习 (Concept Learning)	352
10.7 发现学习 (Discovery as Learning)	357
10.8 小结	358
习题	358
参考文献	360

绪 论

人工智能 (Artificial Intelligence) 是计算机科学、控制论、信息论、神经生理学、心理学、语言学等多种学科互相渗透而发展起来的一门综合性新学科，其诞生可追溯到 50 年代中。1956 年夏季，在美国 Dartmouth 大学，由年青数学助教 J. McCarthy (现斯坦福大学教授) 和他的三位朋友 M. Minsky (哈佛大学年青数学和神经学家，现 MIT 教授)、N. Lochester (IBM 公司信息研究中心负责人) 和 C. Shannon (贝尔实验室信息部数学研究员) 共同发起，邀请 IBM 公司的 T. More 和 A. Samuel、MIT 的 O. Selfridge 和 R. Solomonoff 以及 RAND 公司和 Carnegie 工科大学的 A. Newell 和 H. A. Simon (现均为 CMU 教授) 等人参加夏季学术讨论班，历时两个月。这十位学者都是在数学、神经生理学、心理学、信息论和计算机科学等领域中从事教学和研究工作的学者，在会上他们第一次正式使用了人工智能 (AI) 这一术语，从而开创了人工智能的研究方向。

0.1 人工智能的研究目标

到目前为止，人工智能的发展已走过了 31 年的历程，虽然什么是人工智能，学术界有各种各样的说法和定义，但就其本质而言，人工智能是研究如何制造出人造的智能机器或智能系统，来模拟人类智能活动的能力，以延伸人们智能的科学。至于人类智能活动的能力是什么含义，人们也是有共同认识的。一般地说是人类在认识世界和改造世界的活动中，由脑力劳动表现出来的

能力，更具体一些可概括为：

（1）通过视觉、听觉、触觉等感官活动，接受并理解文字、图象、声音、语言等各种外界的“自然信息”，这就是认识和理解世界环境的能力。

（2）通过人脑的生理与心理活动以及有关的信息处理过程，将感性知识抽象为理性知识，并能对事物运动的规律进行分析、判断和推理，这就是提出概念、建立方法，进行演绎和归纳推理、作出决策的能力。

（3）通过教育、训练和学习过程，日益丰富自身的知识和技能，这就是学习的能力。

（4）对变化多端的外界环境条件，如干扰、刺激等作用能灵活地作出反应，这就是自我适应的能力。

总之，人类智能是涉及信息描述和信息处理的复杂过程，因而实现人工智能是一项艰巨的任务。尽管如此，这门学科还是引起了许多科学和技术工作者的浓厚兴趣，特别是在计算机科学和技术飞速发展和计算机应用日益普及的情况下，许多学者认为实现人工智能的手段已经具备，人工智能开始走向实践阶段。

目前研究人工智能主要有两条途径。一条是心理学家、生理学家们认为大脑是智能活动的物质基础，要揭示人类智能的奥秘，就必须弄清大脑的结构，也就是要从大脑的神经元模型着手研究，搞清大脑信息处理过程的机理，这样人工智能的实现就可迎刃而解。显然由于人脑有上百亿神经元，而且现阶段要进行人脑的物理模拟实验还很困难，因而完成这个任务极其艰巨。但可以看出这一学派是企图创立“信息处理的智能理论”作为实现人工智能的长远研究目标的，这个观点是值得重视的。另一条是计算机科学家们提出的从模拟人脑功能的角度来实现人工智能，也就是通过计算机程序的运行，从效果上达到和人们智能行为活动过程相类似作为研究目标，因而这派学者只是局限于解决“建造智

能机器或系统为工程目标的有关原理和技术”作为实现人工智能的近期目标，这个观点比较实际，目前引起较多人的注意。

总之不论从什么角度来研究人工智能，都是通过计算机来实现，因此可以说人工智能的中心目标是要搞清楚实现人工智能的有关原理，使计算机有智慧、更聪明、更有用。

0.2 人工智能发展简史

1. 萌芽期（1956 年以前）

自古以来，人类就力图根据自己的认识水平和当时的技术条件，企图用机器来代替人的部分脑力劳动，以提高征服自然的能力。公元 850 年，古希腊就有制造机器人帮助人们劳动的神话传说。在我国公元前 900 多年，也有歌舞机器人传说的记载，这说明古代人就有人工智能的幻想。

随着历史的发展，到十二世纪末至十三世纪初年间，西班牙的神学家和逻辑学家 Romen Luee 试图制造能解决各种问题的通用逻辑机。十七世纪法国物理学家和数学家 B.Pascal 制成了世界第一台会演算的机械加法器并获得实际应用。随后德国数学家和哲学家 G.W.Leibniz 在这台加法器的基础上发展并制成了进行全部四则运算的计算器。他还提出了逻辑机的设计思想，即通过符号体系，对对象的特征进行推理，这种“万能 符号”和“推理计算”的思想是现代化“思考”机器的萌芽，因而他曾被后人誉为数理逻辑的第一个奠基人。十九世纪英国数学和力学家 C.Babbage 致力于差分机和分析机的研究，虽因条件限制未能完全实现，但其设计思想不愧为当年人工智能的最高成就。

进入本世纪后，人工智能相继出现若干开创性的工作。1936 年，年仅 24 岁的英国数学家 A.M.Turing 在他的一篇“理想计算

机”的论文中，就提出了著名的图林机模型，1945年他进一步论述了电子数字计算机的设计思想，1950年他又在“计算机能思维吗？”一文中提出了机器能够思维的论述，可以说这些都是图林为人工智能所作的杰出贡献。1938年德国青年工程师 Zuse 研制成了第一台累计数字计算机 Z-1，后来又进行了改进，到 1945 年他又发明了 Planka.kel 程序语言。此外，1946 年美国科学家 J.W.Mauchly 等人制成了世界上第一台电子数字计算机 ENIAC。还有同一时代美国数学家 N.Wiener 控制论的创立，美国数学家 C.E.Shannon 信息论的创立，英国生物学家 W.R.Ashby 所设计的脑等，这一切都为人工智能学科的诞生 作了理论和实验工具的巨大贡献。

2. 形成时期（1956—1961年）

1956 年历史性的聚会被认为是人工智能学科正式诞生的标志，从此在美国开始形成了以人工智能为研究目标的几个研究组：如 Newell 和 Simon 的 Carnegie-RAND 协作组；Samuel 和 Gelernter 的 IBM 公司工程课题研究组；Minsky 和 McCarthy 的 MIT 研究组等，这一时期人工智能的主要研究工作有下述几个方面。

1957 年 A.Newell、J.Shaw 和 H.Simon 等人的心理学小组编制出一个称为逻辑理论机 LT (The Logic Theory Machine) 的数学定理证明程序，当时该程序证明了 B.A.W.Russell 和 A.N.Whitehead 的“数学原理”一书第二章中的 38 个定理（1963 年修订的程序在大机器上终于证完了该书中全部 52 个定理）。后来他们又揭示了人在解题时的思维过程大致可归结为三个阶段：

- (1) 先想出大致的解题计划；
- (2) 根据记忆中的公理、定理和推理规则组织解题过程；

(3) 进行方法和目的分析，修正解题计划。

这种思维活动不仅解数学题时如此。解决其他问题时也大致如此。基于这一思想，他们于1960年又编制了能解十种类型不同课题的通用问题求解程序 GPS(General Problem Solving)。另外他们还发明了编程的表处理技术和 NSS 国际象棋机。和这些工作有联系的 Newell 关于自适应象棋机的论文和 Simon 关于问题求解和决策过程中合理选择和环境影响的行为理论的论文，也是当时信息处理研究方面的巨大成就。后来他们的学生还做了许多工作，如人的口语学习和记忆的 EPAM 模型 (1969年)、早期自然语言理解程序 SAM-CAM 等。此外他们还对启发式求解方法进行了探讨。

1956 年 Samuel 研究的具有自学习、自组织、自适应能力的西洋跳棋程序是 IBM 小组有影响的工作，这个程序可以像一个优秀棋手那样，向前看几步来下棋。它还能学习棋谱，在分析大约 175000 幅不同棋局后，可猜测出书上所有推荐的走步，准确度达 48%，这是机器模拟人类学习过程卓有成就的探索。1959 年这个程序曾战胜设计者本人，1962 年还击败了美国一个州的跳棋大师。

在 MIT 小组，1959 年 McCarthy 发明的表 (符号) 处理语言 LISP，成为人工智能程序设计的主要语言，至今仍被广泛采用。1958 年 McCarthy 建立的行动计划咨询系统以及 1960 年 Minsky 的论文“走向人工智能的步骤”，对人工智能的发展都起了积极的作用。

此外，1956 年 N. Chomsky 的文法体系，1958 年 Selfridge 等人的模式识别系统程序等，都对人工智能的研究产生有益的影响。这些早期成果，充分表明人工智能作为一门新兴学科正在茁壮成长。

3. 发展时期 (1961 年以后)

六十年代以来，人工智能的研究活动越来越受到重视。为了揭示智能的有关原理，研究者们相继对问题求解、博弈、定理证明、程序设计、机器视觉、自然语言理解等领域的课题进行了深入的研究。二十多年来，不仅使研究课题有所扩展和深入，而且还逐渐搞清了这些课题共同的基本核心问题以及它们和其他学科间的相互关系。1974 年 N. J. Nilsson 对发展时期的一些工作写过一篇综述论文，他把人工智能的研究归纳为四个核心课题和八个应用课题，并分别对它们进行了论述。

这一时期中某些课题曾出现一些较有代表性的工作，1965 年 J. A. Robinson 提出了归结（消解）原理，推动了自动定理证明这一课题的发展。70 年代初，T. Winograd、R. C. Schank 和 R. F. Simmons 等人在自然语言理解方面做了许多发展工作，较重要的成就是 Winograd 提出的积木世界中理解自然语言的程序。关于知识表示技术有 C. Green (1966 年) 的一阶谓词演算语句，M. R. Quillian (1966 年) 的语义记忆的网络结构，R. F. Simmons (1973 年) 等人的语义网结构，R. C. Schank (1972 年) 的概念网结构，M. Minsky (1974 年) 的框架系统的分层组织结构等。关于专家系统自 1965 年研制 DENDRAL 系统以来，一直受到人们的重视，这是人工智能走向实际应用最引人注目的课题。1977 年 E. A. Feigenbaum 提出了知识工程 (Knowledge Engineering) 的研究方向，目前已导致专家系统和知识库系统更深入的研究和开发工作。此外智能机器人、自然语言理解和自动程序设计等课题，也是这一时期较集中的研究课题，也取得不少成果。

这一时期学术交流的发展对人工智能的研究有很大推动作用。1969 年国际人工智能联合会成立，并举行第一次学术会议 IJCAI-69 (International Joint Conference on Artificial In-

telligence)，以后每两年召开一次，至今已召开了十次会议。随着人工智能研究的发展，1974年又成立了欧洲人工智能学会，并召开第一次会议 ECAI (European Conference on Artificial Intelligence)，随后也是相隔两年召开一次，至今也开了七次会议。此外许多国家也都有本国的人工智能学术团体。在人工智能刊物方面，1970年创办了《Artificial Intelligence》国际性期刊，爱丁堡大学还不定期出版《Machine Intelligence》杂志，还有IJCAI 会议文集，ECAI 会议文集等。此外 ACM, AFIPS 和 IEEE 等刊物也刊载人工智能的论著。

美国是人工智能的发源地，随着人工智能的发展，世界各国有关学者也都相继加入这一行列，英国在60年代就起步人工智能的研究，到70年代，在爱丁堡大学还成立了《人工智能》系。日本和西欧一些国家虽起步较晚，但发展都较快，苏联对人工智能研究也开始予以重视。我国是从1978年才开始人工智能课题的研究，主要在定理证明、汉语自然语言理解、机器人及专家系统方面设立课题，并取得一些初步成果。近几年国内也相应成立中国人工智能学会、中国计算机学会人工智能和模式识别专业委员会，中国自动化学会模式识别与机器智能专业委员会，开展这方面的学术交流。此外国家也正着手兴建若干个人工智能研究中心实验室，这些都将促进我国人工智能的研究，为这一学科的发展作出贡献。

0.3 人工智能研究的课题

当前人工智能这门学科仍处在蓬勃发展中，并已取得许多研究成果。和其他学科的发展一样，人工智能目前也总结出若干个对实现人工智能系统来说都具有一般意义的核心课题，这就是：

(1) 知识的模型化和表示方法；

- (2) 启发式搜索理论;
- (3) 各种推理方法(演绎推理、规划、常识性推理、归纳推理等);
- (4) 人工智能系统结构和语言。

显然,这些课题的新成果都将进一步推动人工智能应用课题的研究。此外,目前人工智能的研究更多的是结合具体应用领域来进行的。下面就来介绍几个主要的应用领域。

1. 自然语言理解 (Natural Language Understanding)

自然语言是人类之间信息交流的主要媒介,由于人类有很强的理解自然语言的能力,因此互相间的信息交流显得轻松自如。然而目前计算机系统和人类之间的交互几乎还只能使用严格限制的各种非自然语言,因此解决计算机系统能理解自然语言的问题,引起人们的兴趣和重视,在第五代计算机研究中,就把它作为重要研究课题之一。此外实现机器翻译过程中,如果计算机确实会理解一个句子的意义,那么就可能进行释义,从而能较通顺地给出译文。目前人工智能研究中,在理解有限范围的自然语言对话和理解用自然语言表达的小段文章或故事方面的程序系统已有一些进展,但由于理解自然语言涉及对上下文背景知识的处理以及根据这些知识进行推理的一些技术,因此实现功能较强的理解系统仍是一个比较艰巨的任务。

2. 数据库的智能检索 (Intelligent Retrieval from Database)

数据库系统是存储某个学科大量事实的计算机系统,随着应用的进一步发展,存储的信息量愈来愈庞大,因此解决智能检索的问题便具有实际意义。

智能信息检索系统应具有如下的功能:

- (1) 能理解自然语言，允许用自然语言提出各种询问；
- (2) 具有推理能力，能根据存储的事实，演绎出所需的答案；
- (3) 系统拥有一定常识性知识，以补充学科范围的专业知识。系统根据这些常识，将能演绎出更一般询问的一些答案来。实现这些功能要应用人工智能的方法。

3. 专家咨询系统 (Expert Consulting Systems)

所谓专家咨询系统就是一种智能的计算机程序系统，该系统存储有某个专门领域中经事先总结，并按某种格式表示的专家知识（构成知识库），以及拥有类似于专家解决实际问题的推理机制（组成推理系统）。系统能对输入信息进行处理，并运用知识进行推理，做出决策和判断，其解决问题的水平达到专家的水准，因此能起到专家的作用或成为专家的助手。专家系统的开发和研究是人工智能研究中面向实际应用的课题，目前受到极大的重视。已开发的系统数以百计，应用领域涉及化学、医疗、地质、气象、交通、教育、军事等，可以说只要有专家工作的场合，就可以开发专家系统。

目前专家系统主要采用基于规则的演绎技术，开发专家系统的关键问题是知识的表示，应用和获取技术，困难在于许多领域中专家的知识往往是琐碎的，不精确的或不确定的，因此目前研究仍集中在这一核心课题。此外对专家系统开发工具的研制发展也很迅速，这对扩大专家系统的应用范围，加快专家系统的开发过程，将起积极的作用。

4. 定理证明 (Theorem Proving)

数学领域中对臆测的定理寻求一个证明，一直被认为是一项需要智能才能完成的任务。证明定理时，不仅需要有根据假设进

行演绎的能力，而且需要有某些直觉的技巧。例如数学家在求证一个定理时，会熟练地运用他丰富的专业知识，猜测应当先证明哪一个引理，精确判断出已有的哪些定理将起作用，并把主问题分解为若干子问题，分别独立进行求解。因此人工智能研究中机器定理证明很早就受到注视，并取得不少成果。

定理证明的研究在人工智能方法的发展中曾起过重要的作用，例如使用谓词逻辑语言，其演绎过程的形式体系研究，帮助人们更清楚地理解推理过程的各个组成部分。许多其他领域的问题，如医疗诊断、信息检索等也可以应用定理证明的方法，因此机器定理证明的研究具有普遍意义。

5. 博弈 (Game Playing)

博弈被认为是智能的活动，人工智能中主要是研究下棋程序，在六十年代就出现了很有名的西洋跳棋和国际象棋的程序，并达到了大师的水平。博弈问题为搜索策略，机器学习等问题的研究课题提供了很好的实际背景，所发展起来的一些概念和方法对其他人工智能问题也很有用。

6. 机器人学 (Robotics)

随着工业自动化和计算机技术的发展，到六十年代机器人开始进入大量生产和实际应用的阶段。尔后由于自动装配、海洋开发、空间探索等实际问题的需要，对机器人的智能水平提出了更高的要求。特别是危险环境，人们难以胜任的场合更迫切需要机器人，从而推动了智能机器人的研究。

机器人学的研究推动了许多人工智能思想的发展，有一些技术可在人工智能研究中用来建立世界状态的模型和描述世界状态变化的过程。关于机器人动作规划生成和规划监督执行等问题的研究，推动了规划方法的发展。此外由于智能机器人是一个综合

性的课题，除机械手和步行机构外，还要研究机器视觉、触觉、听觉等传感技术，以及机器人语言和智能控制软件等。可以看出这是一个涉及精密机械、信息传感技术、人工智能方法、智能控制以及生物工程等学科的综合技术。这一课题研究有利于促进各学科的相互结合，并大大推动人工智能技术的发展。

7. 自动程序设计 (Automatic Programming)

自动程序设计的任务是设计一个程序系统，它接受关于所设计的程序要求实现某个目标的非常高级的描述作为其输入，然后自动生成一个能完成这个目标的具体程序。在某种意义上来说，编译程序实际上就是去做“自动程序设计”的工作。编译程序是接受一段有关于某件事情的源码说明（源程序），然后转换成一个目标码程序（目的程序）去完成这件事情。而这里所说的自动程序设计相当于一种“超级编译程序”，它要求能对高级描述进行处理，通过规划过程，生成得到所需的程序。因而自动程序设计所涉及的基本问题与定理证明和机器人学有关，要用到人工智能的方法来实现，它也是软件工程和人工智能相结合的课题。

自动编出一份程序来获得某种指定结果的任务同论证一份给定的程序将获得某种指定结果的任务是紧密相关的，前者也称程序综合，后者称为程序验证。许多自动程序设计系统将产生一份输出程序的验证作为额外的收益。

自动程序设计研究的重大贡献之一是把程序调试的概念作为问题求解的策略来使用。实践已经发现，对程序设计或机器人控制问题，先产生一个代价不太高的有错误的解，然后再进行修改的作法，要比坚持要求第一次得到的解就完全没有缺陷的作法，通常效率要高得多。

8. 组合调度问题 (Combinatorial and Scheduling Problems)

有许多实际的问题是属于确定最佳调度或最佳组合的问题，例如旅行商问题就是其中之一。这个问题是要求给推销员确定一条最短的旅行路线，他的旅程是从某一个城市出发，然后遍访他所要访问城市，而且每个城市只访问一次，然后回到出发城市。该问题的一般化提法是：对由几个节点组成的一个图的各条边，寻找一条最小耗费的路径，使得这条路径只对每一个节点穿行一次。

在大多数的这类问题中，随着求解问题规模的增大，求解程序都面临着组合爆炸问题。这些问题中有几个（包括旅行商问题）是属于计算理论家称为 NP- 完全性一类的问题。

他们是根据理论上最佳方法计算出所要求解时间（或步数）的最严重情况来排列不同问题的困难程度。这个时间（或步数）是随着问题大小的某种变量（如旅行商问题中，城市数目就是问题大小的一种变量）而增长。例如问题的困难程度可随问题大小按线性、多项式或指数方式增长。

用现在知道的最佳方法求解 NP- 完全性问题，所花费的时间是随着问题规模增大按指数方式增长，但迄今还不知道是否有更快的方法（如只涉及多项式时间）存在。人工智能学者们曾经研究过若干种组合问题的求解方法，他们的努力主要集中在使“时间-问题大小”曲线的变化尽可能地缓慢，即使它必须按指数方式增长。此外有关问题领域的知识，确实是一些较有效的求解方法的关键因素，为处理组合问题而发展起来的许多方法，对其他组合爆炸不甚严重的问题也是有用的。

9. 感知问题 (Perception Problems)

人工智能研究中，已经给计算机系统装上摄象输入以便能够

“看见”周围的东西，或者装上话筒以便能“听见”外界的声音。视觉和听觉都是感知问题，都涉及到要对复杂的输入数据进行处理。实验表明有效的处理方法要求具有“理解”的能力，而理解则要求大量有关感受到的事物的许多基础知识。

在人工智能中研究的感知过程通常包含一组操作，例如可见的景物由传感器编码，并被表示为一个灰度数值的矩阵，这些灰度数值由检测器加以处理，检测器搜索主要图象的成份，如线段、简单曲线、角等等。这些成份又被处理以便根据景物的表面和形状来推测有关景物三维特征的信息，其最终目标则是利用某个适当的模型来表示该景物。例如一个高层描述组成的模型是：“一座山，山顶上有一棵树，山上牛正在吃草”。

整个感知问题的要点是建立一个精炼的表示来取代难以处理的极其庞大的、未经加工的输入数据，这种最终表示的性质和质量取决于感知系统的目标。例如若颜色是重要的，则系统必须予以重视；若空间关系和变量是重要的，则系统必须给予精确的判断。不同的系统将有不同的目标，但所有的系统都必须把来自输入多得惊人的感知数据压缩为一种容易处理和有意义的描述。

在视觉问题中，感知一幅景物的主要困难是候选描述的数量太多。有一种策略是对不同层次的描述作出假设，然后再测试这些假设，这种假设-测试的策略给这个问题提供了一种方法，它可应用于感知过程的不同层次上。此外假设的建立过程还要求大量有关感知对象的知识。

感知问题除了信号处理技术外，还涉及知识表示和推理模型等一些人工智能技术。

第一章 产生式系统

产生式系统(Production System)是1943年Post提出的一种计算形式体系里所使用的术语，主要是使用类似于文法的规则，对符号串作替换运算。到了60年代产生式系统成为认知心理学研究人类心理活动中信息加工过程的基础，并用它来建立人类认识的模型。到现在产生式系统已发展成为人工智能系统中最典型最普遍的一种结构，例如目前大多数的专家系统都采用产生式系统的结构来建造。

为什么要采用产生式系统作为人工智能系统的主要结构呢？这可以有两点理由：

(1) 用产生式系统结构求解问题的过程和人类求解问题时的思维过程很相象(下面要举例说明)，因而可以用它来模拟人类求解问题时的思维过程。

(2) 可以把产生式系统作为人工智能系统的基本结构单元或基本模式看待，就好像是积木世界中的积木块一样，因而研究产生式系统的基本问题就具有一般意义。

本书前面几章是通过产生式系统来讨论AI中问题求解方法的基本思想，本章将介绍产生式系统的组成、分类及有关基本概念。

1.1 产生式系统的组成部分

一个人工智能产生式系统的基本要素是：一个综合数据库(Globle Database)，一组产生式规则(Set of Rules)和一个

控制系统(Control System)。

综合数据库是人工智能产生式系统所使用的主要数据结构，它用来表述问题状态或有关事实，即它含有所求解问题的信息，其中有些部分可以是不变的，有些部分则可能只与当前问题的解有关。人们可以根据问题的性质，用适当的方法来构造综合数据库的信息。

产生式规则的一般形式为

条件→行动

或 前提→结论

即表示成为

if……then……

的形式。其中左半部确定了该规则可应用的先决条件，右半部描述了应用这条规则所采取的行动或得出的结论。一条产生式规则满足了应用的先决条件之后，就可对综合数据库进行操作，使其发生变化。如综合数据库代表当前状态，则应用规则后就使状态发生转换，生成出新状态。

控制系统或策略 (Control Strategies) 是规则的解释程序。它规定了如何选择一条可应用的规则对数据库进行操作，即决定了问题求解过程的推理路线。当数据库满足结束条件时，系统就应停止运行；还要使系统在求解过程中记住应用过的规则序列，以便最终能给出解的路径。

上述产生式系统的定义具有一般性，它可用来模拟任一可计算过程。在研究人类进行问题求解过程时，完全可用一个产生式系统来模拟求解过程，即可作为描述搜索的一种有效方法。作为人工智能中的一种形式体系，它还具有以下优点：

(1) 适合于模拟强数据驱动特点的智能行为。当一些新的数据输入时，系统的行为就要改变。

(2) 易于添加新规则去适应新的情况，而不会破坏系统的

其他部分。这是由于产生式系统的各组成部分具有相对的独立性，因而便于扩展和修改。

下面举例说明如何用产生式系统来描述或表示求解的问题，即如何对具体的问题建立起产生式系统的描述，以及用产生式系统求解问题的基本思想。

1. 八数码游戏(Eight-Puzzle)

在 3×3 组成的九宫格棋盘上，摆有八个将牌，每一个将牌都刻有1—8中的某一个数码。棋盘中留有一个空格，允许其周围的某一个将牌向空格移动，这样通过移动将牌就可以不断改变将牌的布局。这种游戏求解的问题是：给定一种初始的将牌布局或结构（称初始状态）和一个目标的布局（称目标状态），问如何移动将牌，实现从初始状态到目标状态的转变。问题的解答其实就是给出一个合法的走步序列。

要用产生式系统来求解这个问题，首先必须建立起问题的产生式系统描述，即规定出综合数据库、规则集合及其控制策略。这种把一个问题的叙述转化为产生式系统的三个组成部分，在人工智能中通常称为问题的表示。一般来说一个问题可有多种表示方式，而选择一种较好的表示是运用人工智能技术解决实际问题首先要考虑的，而且要有一定的技巧。

设给定的具体问题如图1.1所示。

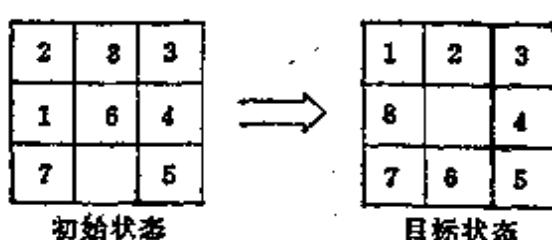


图 1.1 一个八数码游戏实例

(1) 综合数据库：在这里是要选择一种数据结构来表示将牌的布局。通常可用来表示综合数据库的数据结构有符号串、向量、集合、数组、树、表格、文件等。对八数码问题，选用二维数组来表示将牌的布局很直观，

因此该问题的综合数据库可以如下形式表示：

(S_{ij}) , 其中 $1 \leq i, j \leq 3$, $S_{ij} \in \{0, 1, \dots, 8\}$, 且 S_{ij} 互不相等
这样每一个具体的矩阵就可表示一个棋局状态。对八数码游戏，
显然共有 $9! = 362880$ 个状态。所有可能的状态集合就构成该问题
的状态空间或问题空间。可以证明八数码问题的实际问题空间只
有 $\frac{1}{2}9! = 181440$ 。

(2) 规则集合：移动一块将牌（即走一步）就使状态发生
转变。改变状态有4种走法：空格左移、空格上移、空格右移、空
格下移。这4种走法可用4条产生式规则来模拟，应用每条规则都
应满足一定的条件。于是规则集可形式化表示如下：

设 S_{ij} 记矩阵第 i 行第 j 列的数码， i_0, j_0 记空格所在的行、列数
值，即 $S_{i_0 j_0} = 0$ ，则

if $j_0 - 1 \geq 1$ then $S_{i_0 j_0} := S_{i_0 (j_0-1)}$, $S_{(i_0-1) j_0} := 0$;
($S_{i_0 j_0}$ 向左)

if $i_0 - 1 \geq 1$ then $S_{i_0 j_0} := S_{(i_0-1) j_0}$, $S_{i_0 (j_0-1)} := 0$;
($S_{i_0 j_0}$ 向上)

if $j_0 + 1 \leq 3$ then $S_{i_0 j_0} := S_{i_0 (j_0+1)}$, $S_{(i_0+1) j_0} := 0$;
($S_{i_0 j_0}$ 向右)

if $i_0 + 1 \leq 3$ then $S_{i_0 j_0} := S_{(i_0+1) j_0}$, $S_{i_0 (j_0+1)} := 0$;
($S_{i_0 j_0}$ 向下)

(3) 搜索策略：是从规则集中选取规则并作用于状态的一
种广义选取函数。确定某一种策略后，可以算法的形式给出。在
建立产生式系统描述时，还要给出初始状态和目标条件，具体说
明所求解的问题。产生式系统中控制策略的作用就是从初始状态
出发，寻求一个满足一定条件的问题状态。对该八数码问题，初

始状态可表示为 $\begin{pmatrix} 2 & 8 & 3 \\ 1 & 6 & 4 \\ 7 & & 5 \end{pmatrix}$ ；目标描述可表示为 $\begin{pmatrix} 1 & 2 & 3 \\ 8 & & 4 \\ 7 & 6 & 5 \end{pmatrix}$ ，它是一种显

式表示的描述。更一般的情况可以是规定出达到目标的某个真/假条件的描述,如规定要达到第一行将牌数码总和为6的状态。显然这样一个条件,隐含地确定了目标是一个状态的子集——目标集。目标条件也是产生式系统结束条件的基础。

建立了产生式系统描述之后,通过控制策略,可求得实现目标的一个走步序列(即规则序列),这就是所谓的问题的解,如走步序列(上、上、左、下、右)就是一个解。这个解序列是根据控制系统记住搜索目标过程中用过的所有规则而构造出来的。

在一般情况下,问题可能有多个解的序列,但有时会要求得到有某些附加约束条件的解,例如要求步数最少、距离最短等。这个约束条件通常是用耗散或代价(cost)这一概念来概括,这时问题可提为寻找具有最小耗散的解。

现在再来看一下人们是如何来求解八数码游戏的。首先是仔细观察和分析初始的棋局状态,通过思考决定走法之后,就移动某一块将牌,从而改变了布局,与此同时还能判定出这个棋局是否达到了目标。如果尚未达到目标状态,则以这个新布局作为当前状态,重复上述过程一直进行下去,直至到达目标状态为止。可以看出,用产生式系统来描述和求解这个问题,也是在这个问题空间中去搜索一条从初始状态到达某一个目标状态的路径。这完全可以模拟人们的求解过程,也就是可以把产生式系统作为求解问题思考过程的一种模拟。

2. 传教士和野人问题 (Missionaries and Cannibals)

有N个传教士和N个野人来到河边准备渡河,河岸有一条船,每次至多可供k人乘渡。问传教士为了安全起见,应如何规划摆渡方案,使得任何时刻,河两岸以及船上的野人数目总是不超过传教士的数目。即求解传教士和野人从左岸全部摆渡到右岸的过程中,任何时刻满足 M (传教士数) $\geq C$ (野人) 和 $M +$

$C \leq k$ 的摆渡方案。

设 $N=3, k=2$, 则给定的问题可用图1.2表示, 图中 L 和 R 表示左岸和右岸, $B=1$ 或 0 分别表示有船或无船。约束条件是: 两

	L R			L R	
	M	C		M	C
M	3 0			0 3	
C	3 0		↔	0 3	
B	1 0			0 1	

图 1.2 M-C问题实例

岸上 $M \geq C$, 船上 $M + C \leq 2$ 。

(1) 综合数据库: 用三元组表示, 即

(M_L, C_L, B_L) , 其中 $0 \leq M_L, C_L \leq 3$, $B_L \in \{0, 1\}$

此时问题描述简化为

$$(3, 3, 1) \longrightarrow (0, 0, 0)$$

$N=3$ 的M-C问题，状态空间的总状态数为 $4 \times 4 \times 2 = 32$ ，根据约束条件的要求，可以看出只有20个合法状态。再进一步分析后，又发现有4个合法状态实际上是不可能达到的。因此实际的问题空间仅由16个状态构成。下表列出分析的结果：

(M_L, C_L, B_L)	(M_L, C_L, B_L)
$(0 \ 0 \ 1)$ 达不到	$(0 \ 0 \ 0)$
$(0 \ 1 \ 1)$	$(0 \ 1 \ 0)$
$(0 \ 2 \ 1)$	$(0 \ 2 \ 0)$
$(0 \ 3 \ 1)$	$(0 \ 3 \ 0)$ 达不到
$(1 \ 0 \ 1)$ 不合法	$(1 \ 0 \ 0)$ 不合法
$(1 \ 1 \ 1)$	$(1 \ 1 \ 0)$
$(1 \ 2 \ 1)$ 不合法	$(1 \ 2 \ 0)$ 不合法

(1 3 1)	非法	(1 3 0)	非法
(2 0 1)	非法	(2 0 0)	非法
(2 1 1)	非法	(2 1 0)	非法
(2 2 1)		(2 2 0)	
(2 3 1)	非法	(2 3 0)	非法
(3 0 1)	达不到	(3 0 0)	
(3 1 1)		(3 1 0)	
(3 2 1)		(3 2 0)	
(3 3 1)		(3 3 0)	达不到

(2) 规则集合: 由摆渡操作组成。该问题主要有两种操作: p_{mc} 操作 (规定为从左岸划向右岸) 和 q_{mc} 操作 (从右岸划向左岸)。每次摆渡操作, 船上人数有五种组合, 因而组成有 10 条规则的集合。

if $(M_L, C_L, B_L = 1)$ then $(M_L - 1, C_L, B_L - 1)$; (p_{10} 操作)

if $(M_L, C_L, B_L = 1)$ then $(M_L, C_L - 1, B_L - 1)$; (p_{01} 操作)

if $(M_L, C_L, B_L = 1)$ then $(M_L - 1, C_L - 1, B_L - 1)$; (p_{11} 操作)

if $(M_L, C_L, B_L = 1)$ then $(M_L - 2, C_L, B_L - 1)$; (p_{20} 操作)

if $(M_L, C_L, B_L = 1)$ then $(M_L, C_L - 2, B_L - 1)$; (p_{02} 操作)

if $(M_L, C_L, B = 0)$ then $(M_L + 1, C_L, B + 1)$; (q_{10} 操作)

if $(M_L, C_L, B = 0)$ then $(M_L, C_L + 1, B + 1)$; (q_{01} 操作)

if $(M_L, C_L, B = 0)$ then $(M_L + 1, C_L + 1, B + 1)$; (q_{11} 操作)

if $(M_L, C_L, B = 0)$ then $(M_L + 2, C_L, B + 1)$; (q_{20} 操作)

if $(M_L, C_L, B = 0)$ then $(M_L, C_L + 2, B + 1)$; (q_{02} 操作)

(3) 初始和目标状态: 即 $(3, 3, 1)$ 和 $(0, 0, 0)$ 。和八数码游戏的问题一样, 建立了产生式系统描述之后, 就可以通过控制策略, 对状态空间进行搜索, 求得一个摆渡操作序列, 使其能够实现目标状态。

在讨论用产生式系统求解问题时, 有时引入状态空间图的概念很有帮助。状态空间图是一个有向图, 其节点可表示问题的各种状态(综合数据库), 节点之间的弧线代表一些操作(产生式规则), 它们可把一种状态导向另一种状态。这样建立起来的状态空间图, 描述了问题所有可能出现的状态及状态和操作之间的关系, 因而可以较直观地看出问题的解路径及其性质。实际上只有问题空间规模较小的问题才可能作出状态空间图, 例如 $N=3$ 的M-C问题, 其状态空间图如图1.3所示。由于每个摆渡操作都有对应的逆操作, 即 p_{mc} 对应 q_{mc} , 所以该图也可表示成具有双向弧的形式。

从状态空间图看出解序列相当之多, 但最短解序列只有4个, 均由11次摆渡操作构成。若给定其中任意两个状态分别作为初始和目标状态, 就立即可找出对应的解序列来。在一般情况下, 求解过程就是对状态空间搜索出一条解路径的过程。

以上两个例子说明了建立产生式系统描述的过程, 这也就是所谓问题的表示。对问题表示的好坏, 往往对求解过程的效率有很大影响。一种较好的表示法会简化状态空间和规则集表示, 例如八数码问题中, 如用将牌移动来描述规则, 则8块将牌就有32条的规则集, 显然用空格走步来描述就简单得多。又如M-C问题中, 用 3×2 的矩阵给出左、右岸的情况来表示一种状态当然可以,

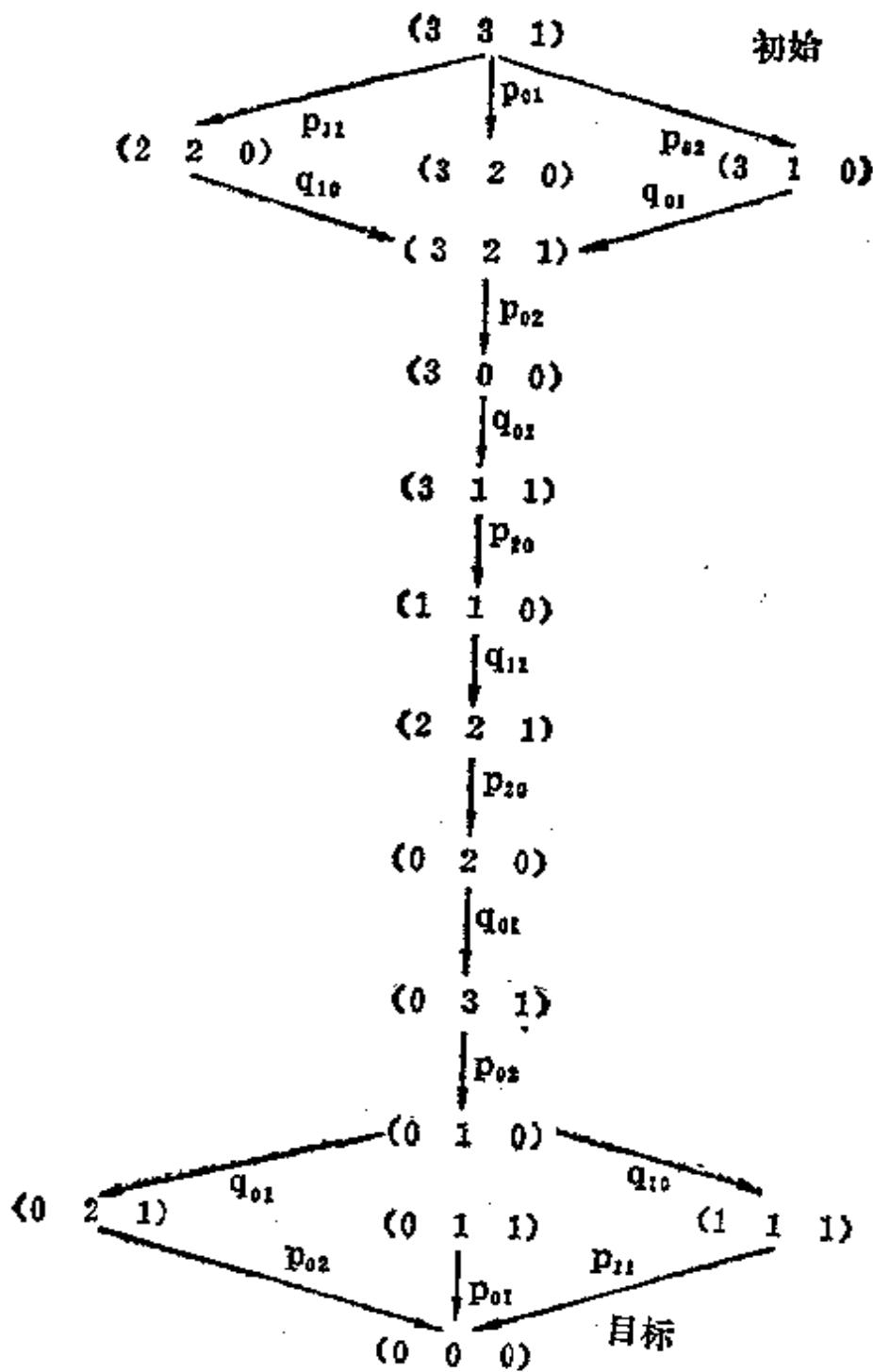


图 1.3 M-C 问题状态空间图

但显然仅用描述左岸的三元组描述就足以表示出整个情况，因此必须十分重视选择较好的问题表示法。以后的讨论还可以看到高效率的问题求解过程与控制策略有关，合适的控制策略可缩小状态空间的搜索范围，提高求解的效率。

1.2 产生式系统的基本过程

用产生式系统求解八数码游戏和M-C问题这一类问题时，其基本算法可写成如下形式：

过程 PRODUCTION

1. DATA \leftarrow —初始数据库
2. until DATA 满足结束条件以前, do:
3. begin
4. 在规则集中，选某一条可应用于 DATA 的规则 R
5. DATA \leftarrow R 应用到 DATA 得到的结果
6. end

这个过程是不确定的，因为在第 4 步没有明确规定如何挑选一条合用的规则，但用它来求解问题，循环过程实际上就是一个搜索过程。下面先简要介绍控制策略的问题，下一章再详细讨论各种搜索策略的具体算法。

1.3 产生式系统的控制策略

上述的 PRODUCTION 过程中，如何选择一条可应用的规则，作用于当前的综合数据库，生成新的状态以及记住选用的规则序列是构成控制策略的主要内容。对大多数的人工智能应用问题，所拥有的控制策略知识或信息并不足以使每次通过算法第 4

步时，一下子就能选出最合适的一条规则来，因而人工智能产生式系统的运行就表现出一种搜索过程，在每一个循环中选一条规则试用，直至找到某一个序列能产生一个满足结束条件的数据库为止。由此可见高效率的控制策略是需要有关被求解问题的足够知识，这样才能在搜索过程中减少盲目性，比较快地找到解路径。

1. 分类

控制策略可划分为两大类：

{ 不可撤回方式 (Irrevocable)
试探性方式 (Tentative) { 回溯方式 (Backtracking)
 图搜索方式 (Graph-search)

下面举例说明这几种策略的基本思想。

(1) 不可撤回方式：这种方式是利用问题给出的局部知识来决定如何选取规则的，就是说根据当前可靠的局部知识选一条可应用规则并作用于当前综合数据库。接着再根据新状态继续选取规则，搜索过程一直进行下去，不必考虑撤回用过的规则。这是由于在搜索过程中如能有效利用局部知识，即使使用了一条不理想的规则，也不妨碍下一步选得另一条更合适的规则。这样不撤消用过的规则，并不影响求到解，只是解序列中可能多了一些不必要的规则。显然这种策略具有控制简单的优点，下面用登山问题来进一步说明这种方式的基本思想。

人们在登山过程中，目标是爬到峰顶，问题就是确定如何一步一步地朝着目标前进达到顶峰。其实这就是一个在“爬山”过程中寻求函数的极大值问题。我们很容易想到利用高度随位置变化的函数 $H(P)$ 来引导爬山，就可实现不可撤回的控制方式。

图1.4表示出登山运动员当前所处某一位置 P_0 ，如果规定好有四种走法[向东(Δx)、向西($-\Delta x$)、向北(Δy)、向南($-\Delta y$)

迈出一步，这相当于有4条规则），那么这时可以用 $H(P)$ 计算一下不同方向时高度变化的情况，即求出 $\Delta z_1 = H(\Delta x) - H_0$ 、 $\Delta z_2 =$

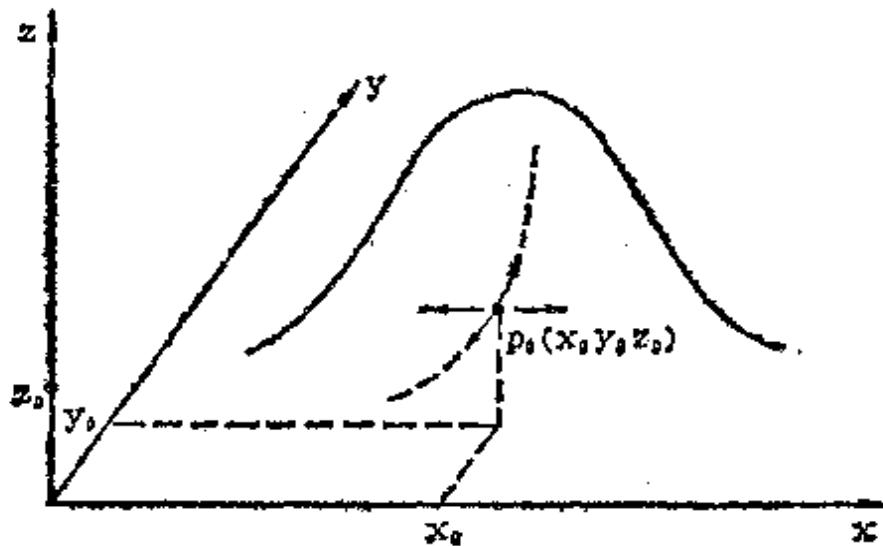


图 1.4 爬山过程示意图

$H(-\Delta x) - H_0$ 、 $\Delta z_3 = H(\Delta y) - H_0$ 、 $\Delta z_4 = H(-\Delta y) - H_0$ ，然后选择 Δz 变化最大的那个走向攀登一步，进入一个新的位置 P ，即在梯度最陡的方向前进一步。然后从 P 开始重复这一过程直至到达某一点为止，这一点就是顶峰点，再进行试探都会导致高度的下降。

用不可撤回的方式（爬山法）来求解登山问题，只有在登单峰的山时才总是有效的（即对单极值的问题可找到解）。对于比较复杂的情况，如碰到多峰、山脊或平顶的情况时，爬山搜索法并不总是有效的。图 1.5 示出这些情况下可能遇到的问题。可以看出，多峰时如果初始点处在非主峰的区域，则只能找到局部优的点上，即得到一个虚假的实现了目标的错觉。对有山脊的情况，如果搜索方向与山脊的走向不一致，则就会停留在山脊处，并以为找到极值点。当出现大片平原区把各山包孤立起来时，就会在平顶区漫无边际的搜索，总是试验不出度量函数有变化的情

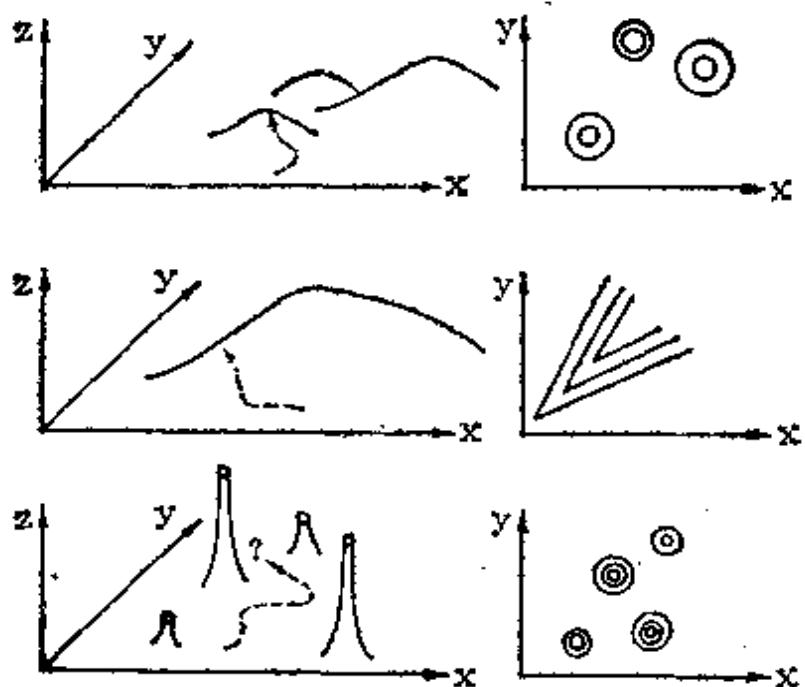


图 1.5 复杂情况出现的问题

况，这导致了随机盲目的搜索。

下面看一看如何运用爬山过程的思想使产生式系统具有不可撤回的控制方式。首先要建立一个描述综合数据库变化的函数，如果这个函数具有单极值，且这个极值对应的状态就是目标，则不可撤回的控制策略就是选择使函数值发生最大增长变化的那条规则来作用于综合数据库，如此循环下去直到没有规则使函数值继续增长，这时函数值取最大值，满足结束条件。

我们举八数码游戏的例子加以说明。用“不在位”将牌个数并取其负值作为状态描述的函数 $-W(n)$ （“不在位”将牌个数是指当前状态与目标状态对应位置逐一比较后有差异的将牌总个数，用 $W(n)$ 表示，其中 n 表示任一状态），例如图1.1中的初始状态，其函数值是 -4 ，而对目标状态，函数值是 0 。用这样定义的函数就能计算出任一状态的函数值来。

从初始状态出发看一看如何应用这个函数来选取规则。对初始状态，有三条可应用规则，空格向左和空格向右这两条规则生成的新状态，其 $-W(n)$ 均为 -5 ，空格向上所得新状态，其 $-W(n) = -3$ ，比较后看出这条规则可获得函数值的最大增长，所以产生式系统就选择这条规则来应用。按此一步步进行下去，直至产生式系统结束时就可获得解。图1.6表示出求解过程所出现的状态序列，图中画圆圈的数字就是爬山函数值。从图中还可看出，沿着状态变化路径，出现有函数值不增加的情况，就是说出现了没有一条合适的规则能使函数值增加，这时就要任选一条函数值不减小的规则来应用，如果不存在这样的规则，则过程停止。

从图1.6中所示的情况来看，用爬山策略（不可撤回）能找到一条通往目标的路径。然而一般说来，爬山函数会有多个局部的极大值情况，这样一来就会破坏爬山法找到真正的目标。例如初始状态和目标状态分别如下：

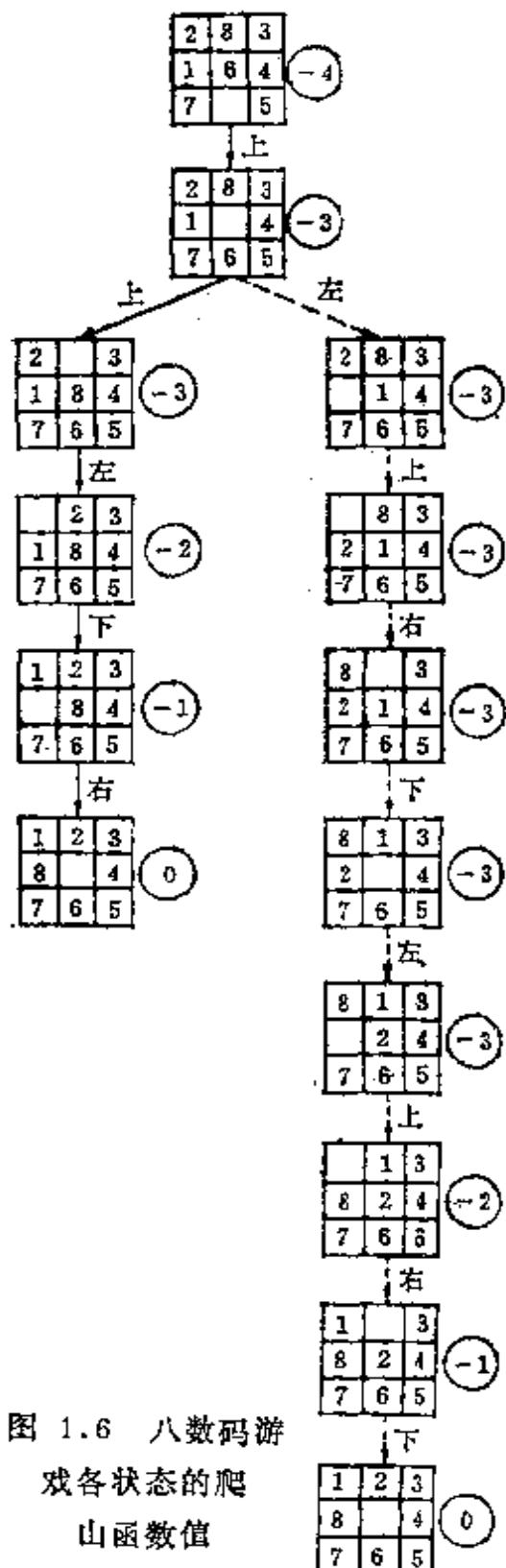


图 1.6 八数码游戏各状态的爬山函数值

$$\begin{array}{ccc}
 1 & 2 & 5 \\
 7 & 4 & \longrightarrow \\
 8 & 6 & 3
 \end{array}
 \qquad
 \begin{array}{ccc}
 1 & 2 & 3 \\
 7 & 4 & \\
 8 & 6 & 5
 \end{array}$$

任意一条可应用于初始状态的规则，都会使 $-W(n)$ 下降，这相当于初始状态的描述函数值处于局部极大值上，搜索过程停止不前，找不到代表目标的全局极大值。

根据以上讨论看出，对人工智能感兴趣的一些问题，使用不可撤回的策略，虽然不可能对任何状态总能选得最优的规则，但是如果应用了一条不合适的规则之后，不去撤消它并不排除下一步应用一条合适的规则，那末只是解序列有些多余的规则而已，求得的解不是最优解，但控制较简单。此外还应当看到，有时很难对给定问题构造出任何情况下都能通用的简单爬山函数（即不具多极值或“平顶”等情况的函数），因而不可撤回的方式具有一定的局限性。

（2）回溯方式：在问题求解过程中，有时会发现应用一条不合适的规则会阻挠或拖延达到目标的过程。在这种情况下，需要有这样的控制策略：先试一试某一条规则，如果以后发现这条规则不合适，则允许退回去，另选一条规则来试。

使用回溯策略首要的问题要研究在什么情况下应该回溯，即要确定回溯条件的问题。其次就是如何利用有用知识进行规则排序，以减少回溯次数。下面我们还用图 1.1 的问题来讨论回溯条件及搜索过程，有关利用知识选取规则的问题留待下一章再讨论，因此应用规则采取事先固定排序依次选取的方式进行，例如以左、上、右、下这种顺序来选取规则。

对八数码游戏，回溯应发生在以下三种情况：①新生成的状态在通向初始状态的路径上已出现过；②从初始状态开始，应用的规则数目达到所规定的数目之后还未找到目标状态（这一组规则的数目实际上就是搜索深度范围所规定的）；③对当前状态，再

没有可应用的规则。

图1.7表示出回溯策略应用于八数码游戏时的一部分搜索图，

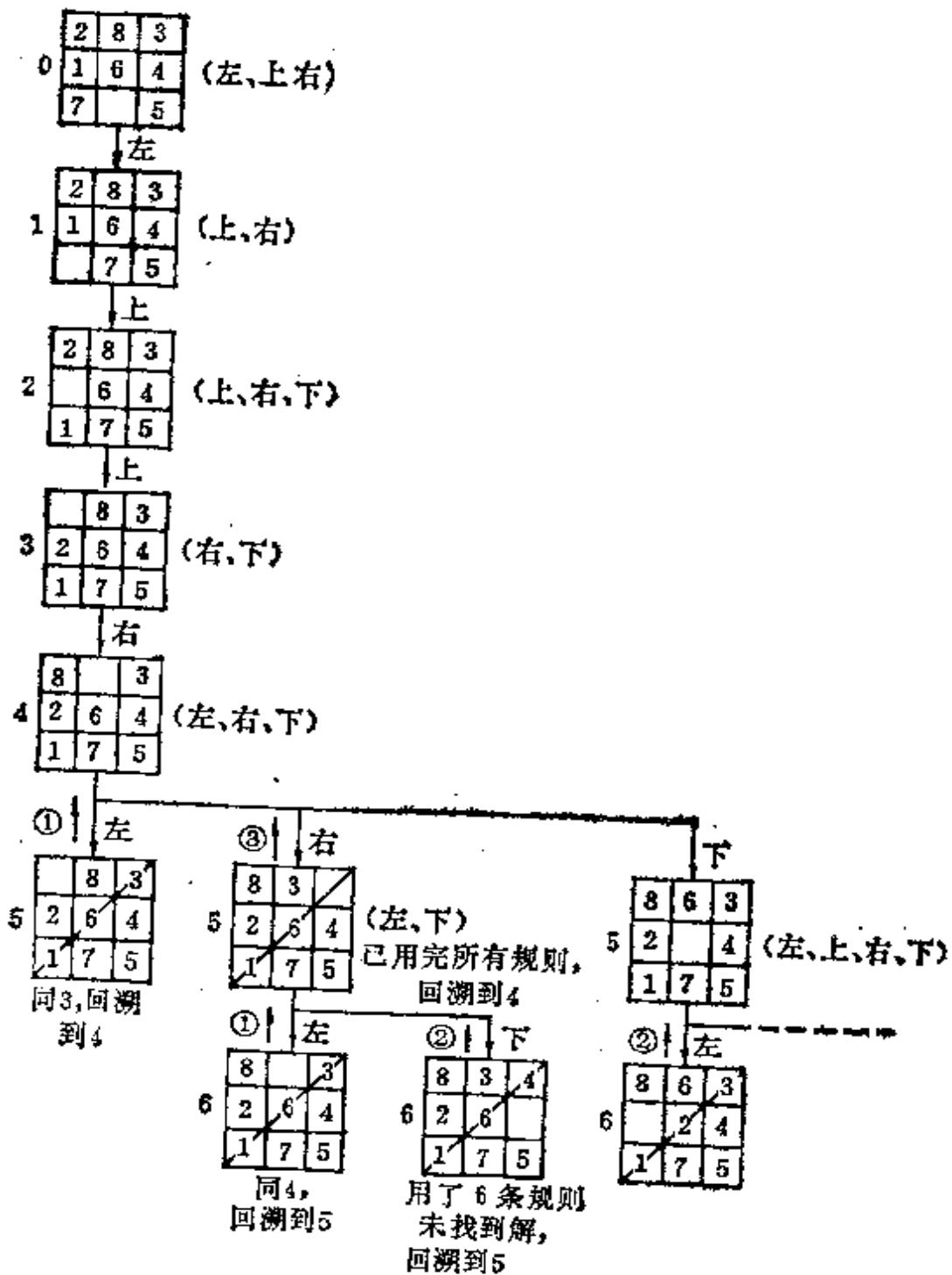


图 1.7 八数码游戏回溯控制方式

这里规定的搜索深度范围到第六层，即用了6条规则后还未找到目标就要回溯到上一层。显然路长为6以内的所有路径都会被搜索到，因此对这个问题，一定能找到解。然而对一般情况，深度设置太浅时，有可能找不到解，设置太深有可能导致回溯次数激增，因而应根据实际情况来规定搜索范围，先设置适中的深度搜索，失败时再逐步加深。

回溯过程是一种可试探的方法，从形式上看不论是否存在对选择规则有用的知识，都可以采用这种策略。即如果没有有用的知识来引导规则选取，那么规则可按任意方式（固定排序或随机）选取；如果有好的选择规则的知识可用，那么用这种知识来引导规则选取，就会减少盲目性，降低回溯次数，甚至不回溯就能找到解，总之一般来说有利于提高效率。此外由于引入回溯机理，可以避免陷入局部极小值的情况，继续寻找其他达到目标的路径。

（3）图搜索方式：如果把问题求解过程用图或树的这种结构来描述，即图中的每一个节点代表问题的状态，节点间的弧代表应用的规则，那么问题的求解空间就可由隐含图来描述。图搜索方式就是用某种策略选择应用规则，并把状态变化过程用图结构记录下来，一直到得出解为止，也就是从隐含图中搜索出含有解路径的子图来。

图1.8表示出一种图搜索策略求解八数码问题时所得到的搜索树。可以看出这是一种穷举的方式，对每一个状态可应用的所有规则都要去试，并把结果记录下来。这样，求得一条解路径要搜索到较大的求解空间。当然，如果利用一些与问题有关的知识来引导规则的选择，有可能搜索较窄的空间就能找到解，这些问题将在第二章研究具体图搜索算法时再进一步讨论。

以上简单介绍了三种控制方式，可以看出不可撤回方式相当于沿着单独的一条路向下延伸搜索下去；回溯方式则不保留完整

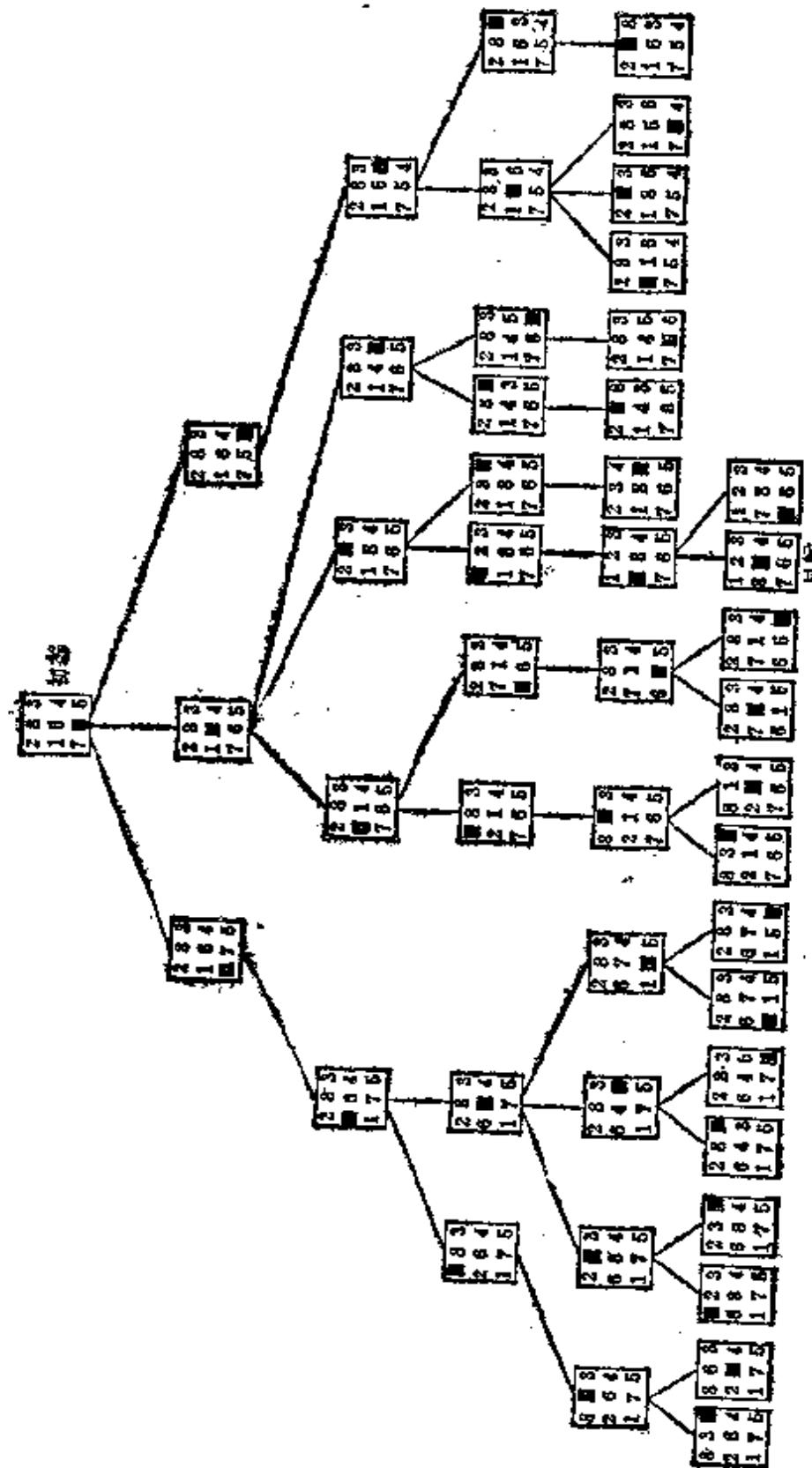


图 1.8 八数码问题的搜索树

的搜索树结构,只记住当前工作的一条路径,回溯就是对这条路径进行修正;图搜索方式则记下完整的搜索树。对一个要求解的具体问题,有可能用不同的方式都能求得解,至于选用哪种方式更适宜,往往还需要根据其他一些实际的要求考虑决定。

1.4 问题的表示

上一节的讨论可以看出高效率的求解过程与有效的控制策略紧密相关,并指出过问题的表示(即综合数据库和规则集的描述)往往对求解时耗费的工作量也有很大的影响。有许多似乎很难的问题,当表示得当时,就可具有较简单的状态描述;因此好的表示是很有意义的。但是一种好的表示有时候是在求解问题过程中取得经验之后产生了新想法才能提出来的,如发现对称关系可以利用,或者发现若干条规则可合并成宏规则等等。下面再讨论两个其他类型的例子来说明用产生式求解时应如何表示。

1. 旅行商问题

一个推销员要到几个城市去办理业务,城市间里程数已知,问题的提法是:从某个城市出发,每个城市只允许访问一次,最后又回到原来的城市,求一条最短距离的路径。

图 1.9 表示出五城市旅行商问题的地图,求从 A 出发经 B、C、D、E 再回到 A 的最短路径。

若每个城市用一个字母表示,则综合数据库可用一个字母组成的表或字符串来表示,如 (A) 表示初始状态, (A × × × × A) 表示目标状态, (A × ×) 表示访问两个城市后的当前状态。

规则集相当于一系列决策: (1) 下一步走向城市 A; (2) 下一步走向城市 B; …… (5) 下一步走向城市 E。对当前的状态,只要某一条规则作用之后能生成出合法的新状态,

那么这一条规则就是可应用规则。例如下一步走向城市A这条规则就不适用于所有城市尚未完全出现的综合数据库。

图 1.10 表示出求解该问题时，用图搜索控制方式可能生成的部分搜索树，树枝上的数字是里程增量，由此可计算出整个旅程的距离来。

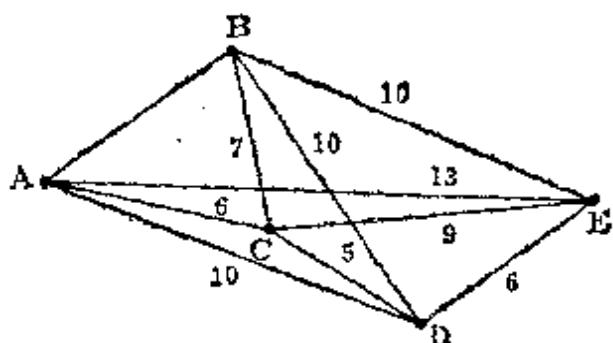


图 1.9 旅行商问题的地图

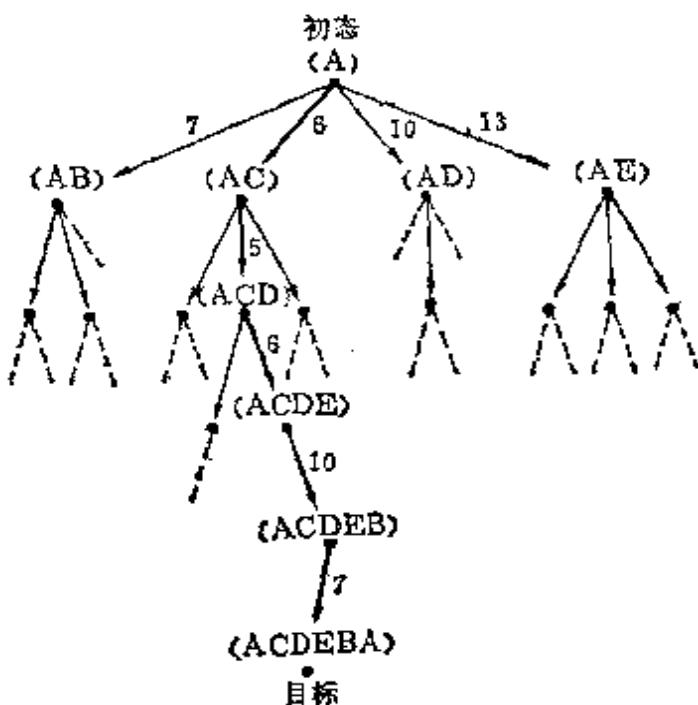


图 1.10 五城市旅行商问题的部分搜索树

2. 句法分析问题

在语言学中，决定任意一个符号序列是不是一个句子的问题，是一个句法分析问题，我们可以用产生式系统来分析这个问题。下面先定义一个分析有限个英语句子的与上下文无关的简单

文法来讨论这个问题。

设该文法有如下的终结符和非终结符：

终结符号： of approves new president company sale
the

非终结符号： S NP VP PP P V DNP DET A N

文法的重写规则是：

N→NP

A NP→NP

DET NP→DNP

P DNP→PP

DNP PP→DNP

V DNP→VP

DNP VP→S

of→P

approves→V

new→A

President→N

company→N

sale→N

the→DET

现在来分析以下的符号串是否属于该语言中的一个句子：

The president of new company approves the sale

综合数据库规定为由若干符号序列构成，初始数据库就是上面给出待分析的符号串。产生式规则可从上述文法重写规则那里推得，一条文法规则右边的符号，可替代综合数据库中与文法规则左边匹配的任一符号串。例如 DNP VP→S 可以把综合数据库中任一 DNP VP 子串用 S 替代，因此可能会有不同的替代结果。当然不含有这样的子串，规则就不能应用。产生式的目标条

件是具有单个符号 S 构成的数据库。图 1.11 表示出这个问题的部分搜索树。

1.5 产生式系统的类型

1. 正向、逆向、双向产生式系统

用产生式系统求解某一个问题时，如果按照规则使用的方式或者说按推理方向来划分的话，则有正向、逆向和双向产生式系统这三种称呼。正向产生式系统是从初始状态出发朝着目标状态这个方向来使用规则，即正推的方式工作的，我们称这些规则为 F 规则。反过来如果选取目标描述作为初始综合数据库逆向进行求解，即逆向使用规则，产生子目标状态，反方向一步一步朝着初始状态方向求解，即逆推方式工作，则称为逆向产生式系统。逆向应用的规则称 B 规则（从子目标应用对应的 F 规则，又立即到达目标状态）。若以双向搜索的方式（即正向和逆向同时进行）去求解问题，则称为双向产生式系统。这时必须把状态描述和目标描述合并构成综合数据库，F 规则只适用于状态描述部分，B 规则则用于目标描述部分。这种类型的搜索，其控制策略所使用的结束条件要表示成综合数据库中状态描述部分与目标描述部分之间某种形式的匹配条件，而且搜索时还要决定每一

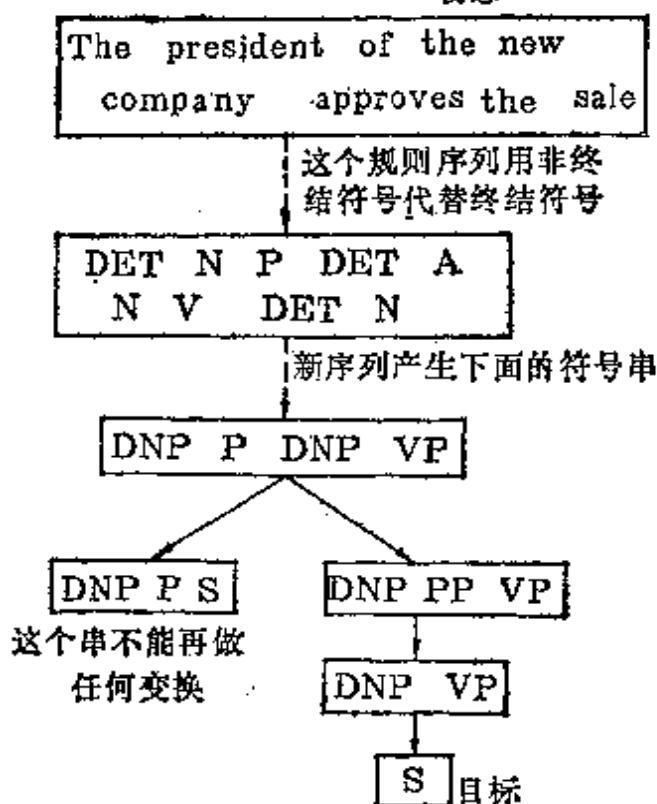


图 1.11 句法分析问题的搜索树

段上要选用 F 规则还是 B 规则。

对于八数码问题，如果只有一个初始状态和一个目标状态，那么正向和逆向求解并没有什么差别，所花的求解工作量也是一样的。但是，当有多个显式表示的目标状态和一个初始状态时，逆向求解有可能具有较低的效率，因为一开始不知道选哪一个目标状态开始求解更好，只能从全部的目标状态集开始搜索。双向求解时，只有当综合数据库中正推状态描述和逆推子目标描述完全匹配时，才结束搜索找到解，可想而知不能及时得到匹配将导致效率降低。

2. 可交换的产生式系统

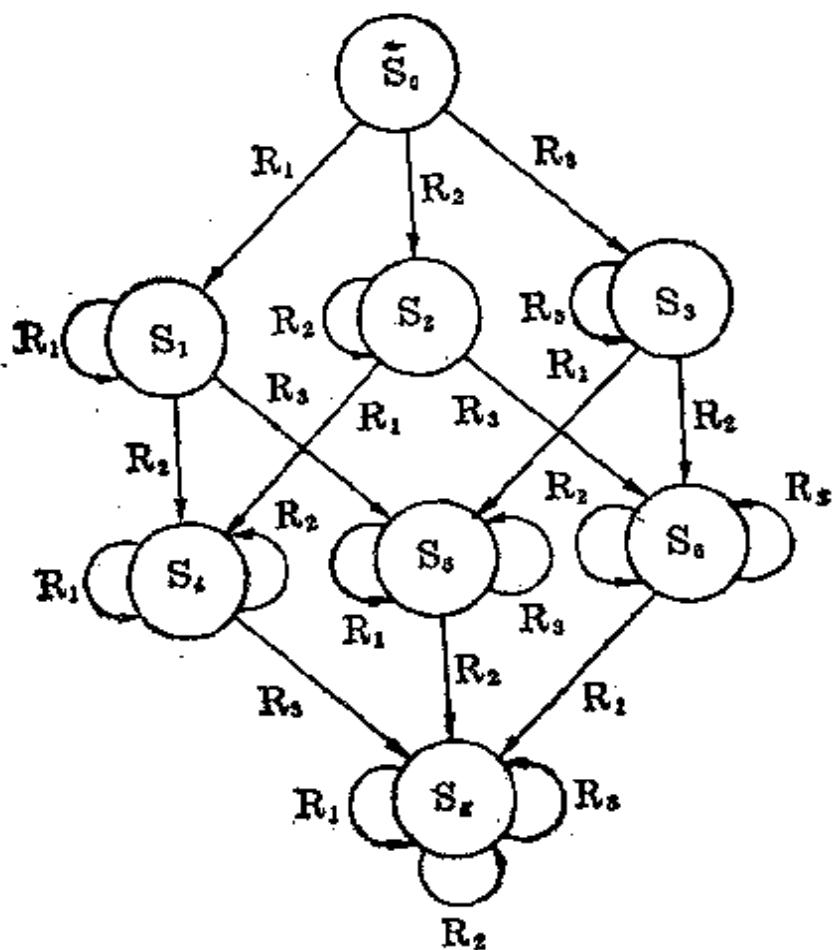


图 1.12 简单的状态空间图

下面分析一下图1.12所示的一个简单状态空间图。图中 S_0 是初始状态， S_n 是目标状态，可应用于 S_0 的规则集 $\{R_1, R_2, R_3\}$ 对状态空间其他任一状态都可以应用，也就是对 S_0 生成的新状态 S_1, S_2, S_3 ，这三条规则仍然适用，同样对它们的后裔也是这样。此外，还可以看出从 S_0 到达目标 S_n 解序列与这三条规则的使用次序无关。有这些性质的系统称为可交换产生式系统，可交换性是指这种情况下这几条规则可以任意交换次序而不影响求解。但要注意并不是所使用的整个规则序列可以重新排列，只有那些最初可应用于初始数据库的规则才可交换，而对于生成的数据库所添加的其他可应用规则，则不能随意交换。

一般来说，当一个产生式系统对任何一个数据库 D 都具有如下性质时，这个产生式系统是可交换的：

(1) 可应用于 D 的规则集合，对用了其中任意一条规则之后所生成的任何数据库，这个规则集合还适用；

(2) 满足目标条件的某个数据库 D ，当应用任何一个可应用于数据库 D 的规则之后所生成的任何数据库，仍然满足目标条件；

(3) 若对 D 应用某一规则序列之后得到一个数据库 D' （设有一对应于 $D \rightarrow D'$ 的一条解路），则当改变 D 的可应用规则集合中的规则次序后，仍然可求得解，即求得 D' 与使用满足 D 的可应用规则集合中的规则次序无关。

下面分析一个可交换性的简例。给定一个整数集合 $\{a, b, c\}$ ，可通过把集合中任意一对元素的乘积作为新元素添加到集合中的办法来扩大该整数集，要求通过若干次操作后能生成出所需的整数集合来。用产生式系统求解这个问题，其综合数据库就可用集合表示，则问题的初始状态为 $\{a, b, c\}$ ，设目标条件为具有 a, b, c, ab, bc, ca 这六个元素组成的集合，初始状态可应用的规则集合为

$$\left\{ \begin{array}{ll} R_1: & \text{if } \{a, b, c\} \text{ then } \{a, b, c, ab\} \\ R_2: & \text{if } \{a, b, c\} \text{ then } \{a, b, c, bc\} \\ R_3: & \text{if } \{a, b, c\} \text{ then } \{a, b, c, ca\} \end{array} \right.$$

显然，这个产生式系统具有上述三个性质，具有可交换性，其部分状态空间图如图 1.13 所示。

由于具有可交换性，求解时只需搜索其中任意一条路径，只要解存在就一定能找到目标，不必探索多条路径，因此不可撤回的控制方式在这种系统中使用很合适，因解与最初可应用的规则的次序无关，系统不必提供特殊选择规则的机理。

3. 可分解的产生式系统

我们来研究一个重写问题的产生式系统，其初始数据库为 (C, B, Z) ，产生式规则的依据是如下的重写规则：

$$\begin{aligned} R_1: \quad & C \rightarrow (D, L) \\ R_2: \quad & C \rightarrow (B, M) \\ R_3: \quad & B \rightarrow (M, M) \\ R_4: \quad & Z \rightarrow (B, B, M) \end{aligned}$$

结束条件是生成出只包含 M 组成的数据库，即 (M, \dots, M) 。

用图搜索方式求解这个问题时，搜索得到的部分状态空间图如图 1.14 所示。图中只给出两条达到目标的路径和一条失败的路径。实际搜索时有可能去探索更多的路径，往往导致效率降低。

对于这个问题，为了避免搜索多余的路径，可将初始数据库分解成几个可以独立加以处理的分量，分别对它们进行求解。即可分别对每一个分量数据库，测试产生式规则可应用的条件，然后应用于每一个分量生成出新的数据库，如此分解、生成交替进行下去，直到分量数据库满足某种结束条件为止。要注意一般结束条件也应分解为对应于分量数据库所使用的结束条件。

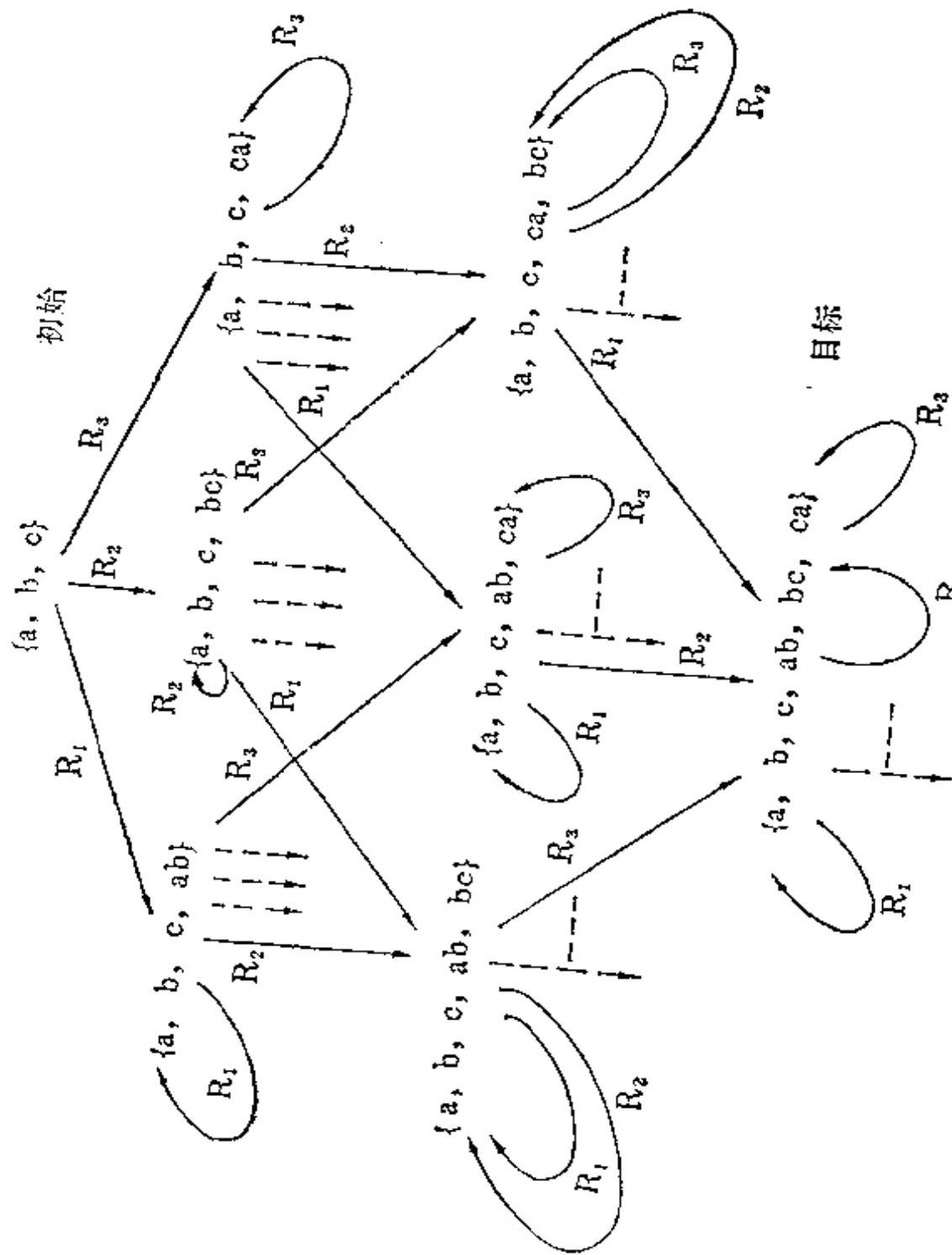


图 1.13 整数集合生成问题的部分状态空间图

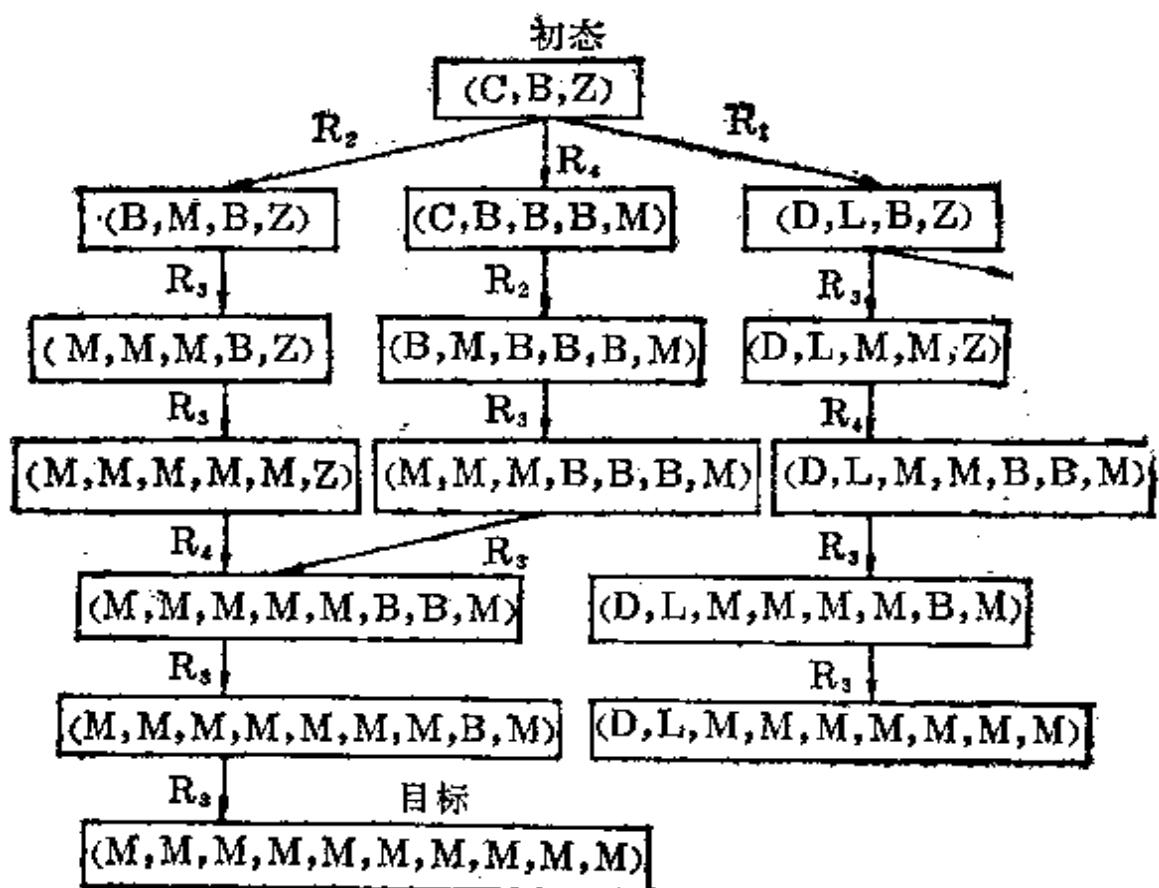


图 1.14 重写问题部分搜索图

能够分解产生式系统的综合数据库和结束条件的产生式系统称为可分解的产生式系统。一个可分解的产生式系统，其基本过程描述如下：

过程 SPLIT

- (1) DATA:=初始数据库
- (2) $\{D_i\}:=$ DATA 的分解式；每个 D_i 元素都看成单独的数据库
- (3) Until $\{D_i\}$ 的所有元素都满足结束条件之前， do;
- (4) begin
- (5) 从 $\{D_i\}$ 中选一个不满足结束条件的 D^*

- (6) 从 $\{D_i\}$ 中删去 D^*
- (7) 在规则集中选择一条可应用于 D^* 的规则 R
- (8) $D := R$ 应用于 D^* 的结果
- (9) $\{d_i\} := D$ 的分解式
- (10) 在 $\{D_i\}$ 上添加 d_i
- (11) end

SPLIT 的控制策略是要在第 5 步选一个分量数据库 D^* ，在第 7 步选一条可应用于 D^* 的规则 R ，显然为满足结束条件，在 $\{D_i\}$ 中不满足结束条件的元素，最终都必须选择到，而对任一选出的 D^* ，可以只考虑选择一条可应用的规则使用。至于分量数据库具体排序方法以及处理分量数据库时规则选取的策略，一般应根据具体问题进行考虑。当采用图搜索方式求解可分解产生式系统时，用一种与或图（树）结构来表示是很清晰的。图 1.15 给出上述重写问题的与或树表示，它和一般的有向图类似，图中的节点代表综合数据库，有向弧组（有小段圆弧链接）表示复合数据库和分解后的各分量数据库之间具有的与关系。即其后继节点（分量数据库）的集合中全部分量都处理完毕，复合数据库才算处理完，我们称这些后继节点为与节点。同样其余有向弧可表示某个分量数据库和应用规则之后产生的新数据库之间具有的或关系，因为在这几个后继节点中（即新数据库，又可能再被分解）只要有一个处理完毕，这个分量数据库就算处理完，我们称这些后继节点为或节点。此外图中双线框表示的那些节点称为终节点，这些节点代表的分量数据库满足结束条件（符号 M ）。

重写问题的解可以用与或图中某一子图来表示，图 1.15 中粗线弧表示出的那一部分子图就代表一个解图，其中“端节点”所对应的数据库，每一个都满足结束条件。解图的一般搜索算法第三章再详细讨论，下面讨论两个应用可分解产生式系统求解问题的实例。

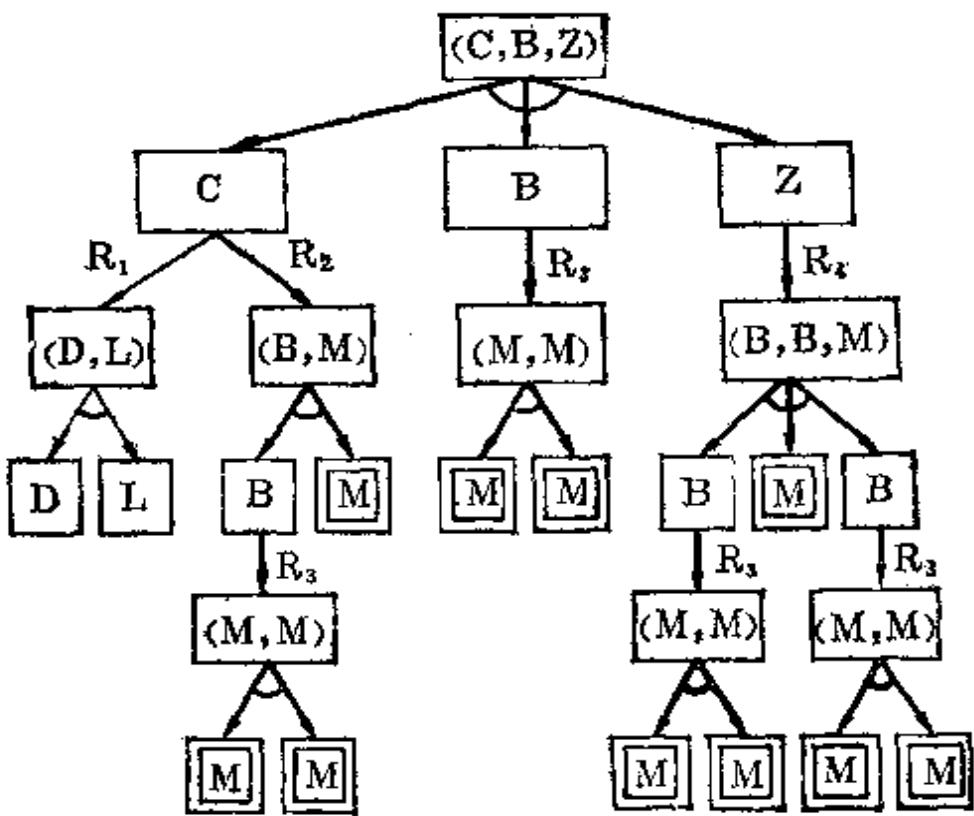


图 1.15 重写问题的与或树

(1) 化学结构生成问题

有机化学中，在已知某一种复杂有机化合物的实验数据条件下（如样品的质谱图），如何确定其结构是实际要解决的问题。这个工作目前已能由著名的 DENDRAL 专家系统（1965）来做。下面只是用简单的碳氢化合物作例子，对该系统中如何生成候选结构这个局部问题的基本思想作一简要说明。

生成候选结构的问题可用产生式系统来描述。综合数据库可表示化合物的公式或“部分结构”，产生式系统规则对该数据库进行操作，使其不断增加结构化程度。例如一开始综合数据库只包含化学公式，中间阶段数据库描述化合物的某种结构，过程结束

时，数据库才包含化合物整个结构的表达式。由于数据库可以分解为若干部分，其中有一些是原化合物中一部分非结构的化学公式，另一些是“部分结构”。产生式规则是把非结构的化学公式转换为部分结构的数据库，即规则集由一组“提出结构”的规则组成。而不包括非结构公式的任一数据库都满足结束条件。

图 1.16 表示化学结构问题求解过程的一部分与或图表示。初始数据库是简单的公式 C_5H_{12} ，图中凡是用两个短竖线括起来的公式都是非结构的，要通过运用“提出结构”的规则，增加其结构化程度，例如有如下规则：

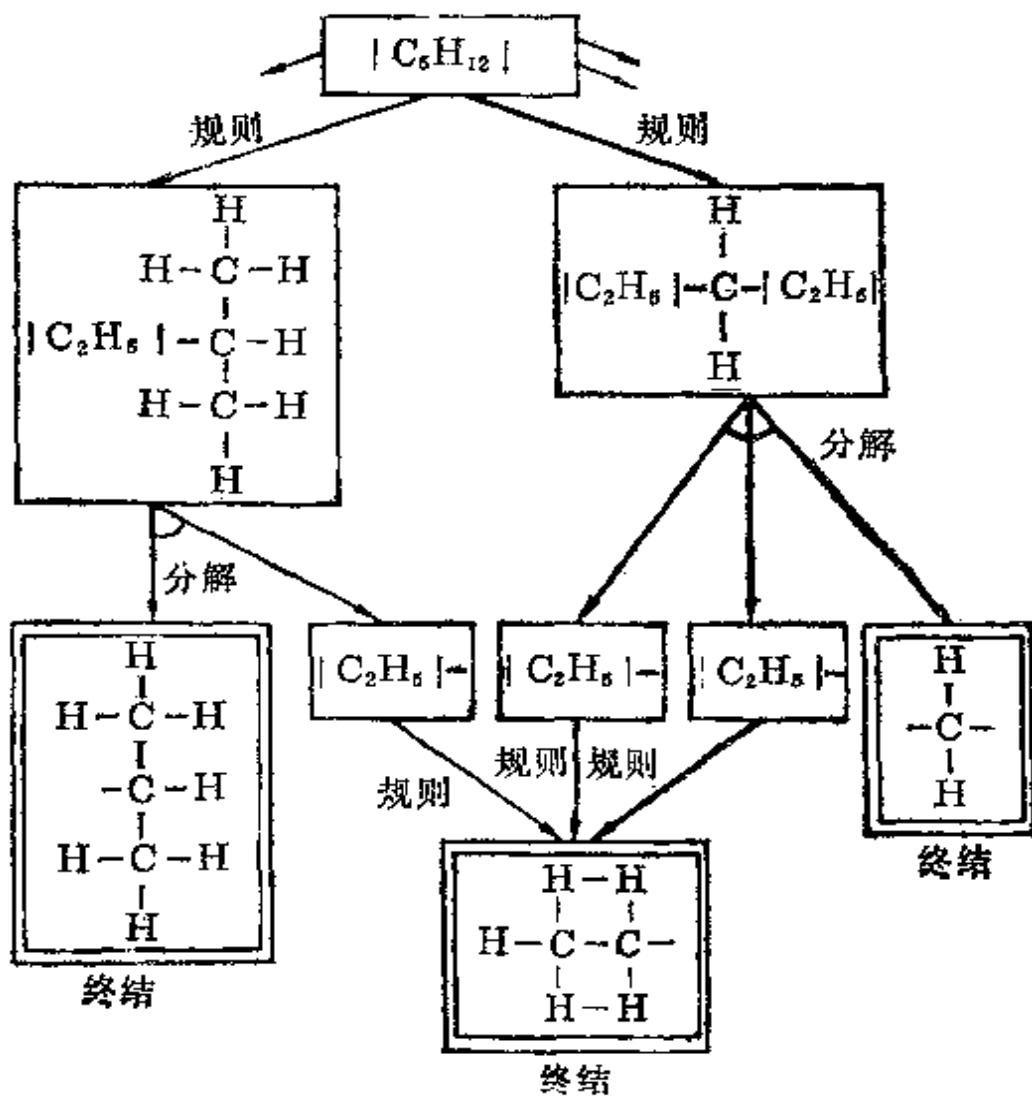
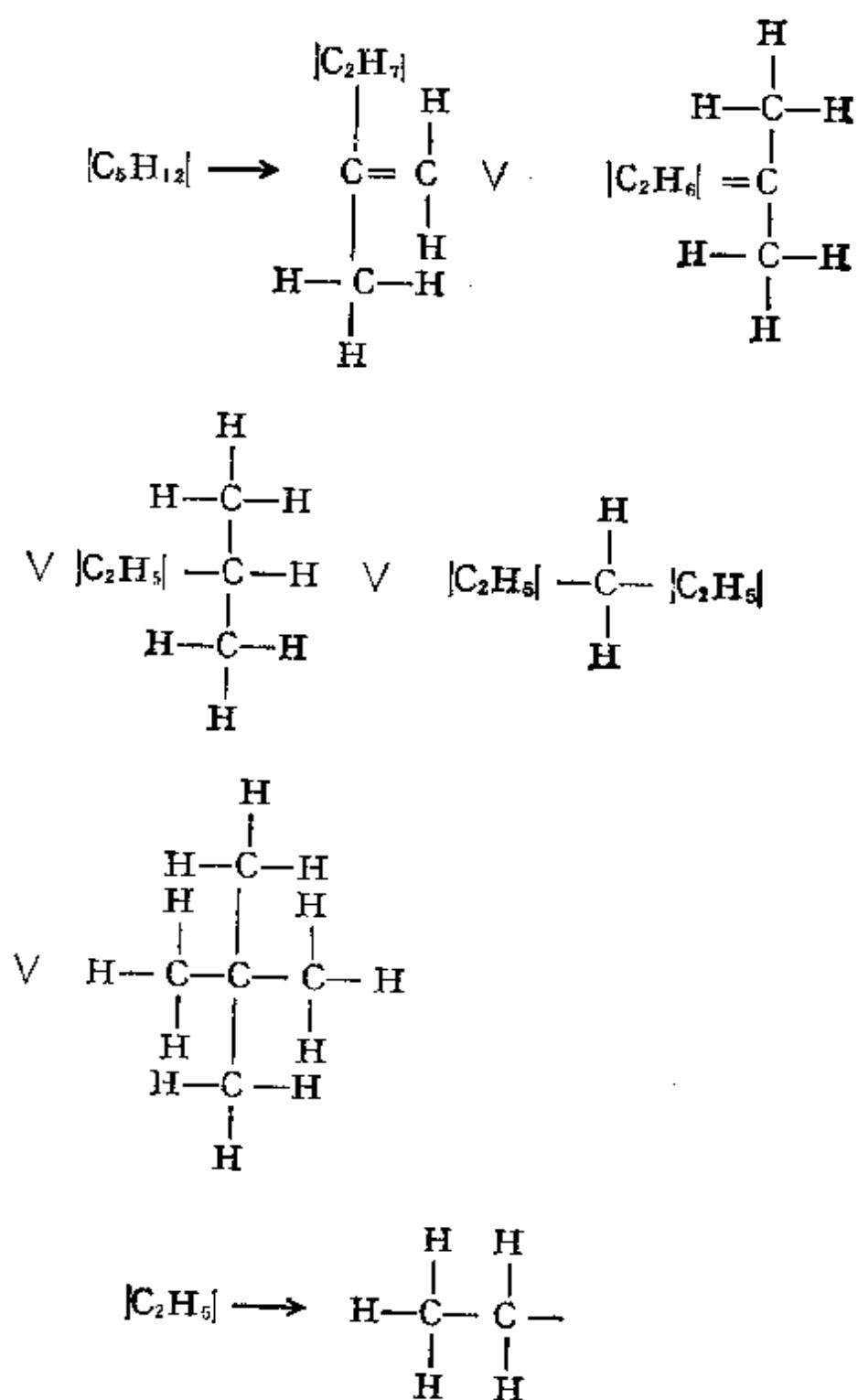
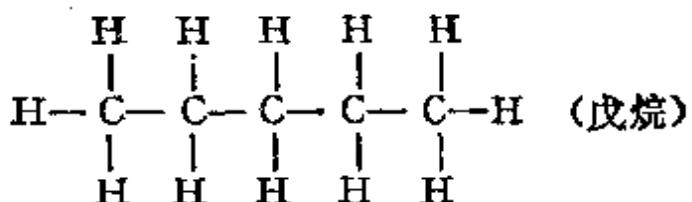


图 1.16 化学结构问题的与或图



图中的每一个解图都对应于一个候选结构，其中粗线表示的一个解图对应如下的化学结构：



(2) 符号积分问题

符号积分的问题是求解不定积分原函数的问题。人工演算是靠运用各种变换公式（如代数、三角变换等）和简单的积分公式以及查找积分表等过程求得相应的原函数。SAINT (1963) 符号积分系统是一个产生式求解系统，不定积分表达式就作为综合数据库描述，产生式规则集由分部积分规则、和的积分分解规则以及其他变换规则（如代数替换和三角替换等）组成，如

$$\int u dv \rightarrow u \int dv - \int v du \quad (\text{分部积分规则})$$

$$\int (f(x) + g(x)) dx \rightarrow \int f(x) dx + \int g(x) dx \quad (\text{和式分解规则})$$

$$\int kf(x) dx \rightarrow k \int f(x) dx \quad (\text{因子规则})$$

其他规则举例

$$\int \frac{x^2 dx}{(2+3x)^{2/3}} \rightarrow \int \frac{1}{9} (z^6 - 4z^3 + 4) dz \quad \text{其中 } z^3 = (2+3x)^{2/3}$$

(代数替换)

$$\int \frac{dx}{x^2 \sqrt{25x^2 + 16}} \rightarrow \int \frac{5}{16} \cot \theta \csc \theta d\theta \quad \text{其中 } x = -\frac{4}{5} \tg \theta$$

(三角替换)

$$\int \frac{z^4 dz}{z^2 + 1} \rightarrow \int \left(z^2 - 1 + \frac{1}{1+z^2} \right) dz \quad (\text{相除化简变换})$$

$$\int \frac{dx}{(x^2 - 4x + 13)^2} \rightarrow \int \frac{dx}{[(x-2)^2 + 9]^2} \quad (\text{配方变换})$$

积分表给出了产生式系统的结束条件，如

$$\int u^n du = \frac{u^{n+1}}{n+1} \quad (n \geq 1)$$

$$\int \sin u du = -\cos u$$

$$\int a^u du = a^u \log a$$

由于任意一个含有积分和式的表达式均可被分解为若干个单独的积分式,每一个积分式又可单独进行求解,因此可以看成是一个可分解的产生式系统。图 1.17 是求解不定积分 $\int \frac{x^4}{(1-x^2)^{5/2}} dx$

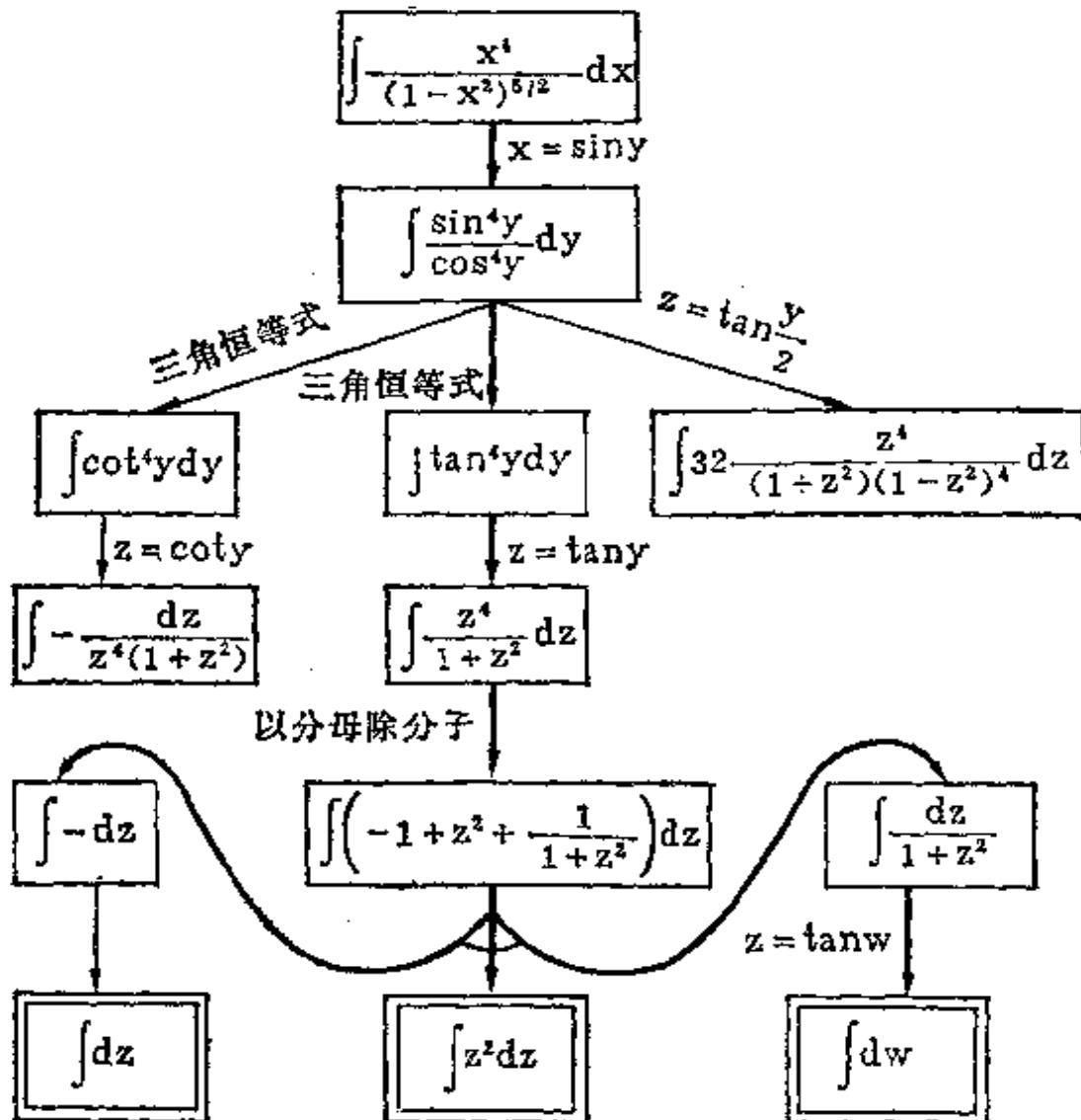


图 1.17 符号积分问题与或树

时所表示的一个与或树,粗线所示的子树是一棵解树,其终节

点都是积分表中的公式，满足产生式系统的结束条件。根据积分表公式可计算出如下结果：

$$\arcsinx + \frac{1}{3}\operatorname{tg}^3(\arcsinx) - \operatorname{tg}(\arcsinx)^3。$$

1.6 小 结

1. 本章一开始就指出产生式系统是AI系统最常见的一种结构，因此分析产生式系统的组成部分及其建立问题是一个很基本的问题。当给定的问题要用产生式系统求解时，要求能掌握建立产生式系统形式化描述的方法，所提出的描述体系应具有一定的一般性，能推广应用于这一类问题更复杂的情况。一般化的产生式系统可用来描述许多重要人工智能系统的工作原理。

2. 产生式系统的综合数据库是指对问题状态的一种描述，这种描述必须便于在计算机中实现，因此它实际上就是AI系统中所使用的数据结构，不同于软件工程中数据库这一术语的概念。计算机程序设计中常用的数据结构类型，如向量、集合、数组、符号串、树、表格乃至文件都可以选作为综合数据库。

3. 问题的表示是研究形式化表示方法的问题，好的问题表示方法既简单又能反映问题的本质，也有利于提高求解的效率。一个问题可有多种的表示方法，当然同一个问题有时也可用不同类型的产生式系统求解。

4. 本章讨论了两个基本过程——过程 PRODUCTION 和过程 SPLIT，它们分别对应一般产生式系统和可分解产生式系统这两种求解方法。PRODUCTION 相当于对问题直接进行求解，而 SPLIT 则是对问题进行归约求解，即通过归约技术，将问题的目标用子目标集合代替，使得如果子目标可得到求解，则主目标也就能解决了，因而归约就是把问题化简到最原本的问题。

的求解方法。此外无论哪一种类型的产生式系统，都可以用正向或逆向的方式求解，不可撤回或试探性的求解策略也都可以使用，实际上要根据具体问题的特点，选用较合适的方法进行求解。

5. 高效能的 AI 系统需要问题领域的知识，通常可把这些知识细分为三种基本类别：陈述性知识是关于表示综合数据库的知识，如待求解问题的特定事实等；过程性知识是关于表示规则部分的知识，如该领域中处理陈述知识所使用的规律性知识；控制知识是关于表示控制策略方面的知识，包括协调整个问题求解过程中所使用的各种处理方法，搜索策略，控制结构有关的知识。用产生式系统求解问题时的主要任务就是如何把问题的知识组织成陈述、过程和控制这三种组成部分，以便在产生式系统中更充分地得到应用。

习 题

1.1 对 $N=5$ 、 $k \leq 3$ 时，求解传教士和野人问题的产生式系统各组成部分进行描述（给出综合数据库、规则集合的形式化描述，给出初始状态和目标条件的描述），并画出状态空间图。

1.2 对量水问题给出产生式系统描述，并画出状态空间图。

有两个无刻度标志的水壶，分别可装 5 升和 2 升的水。设另有一水缸，可用来向水壶灌水或倒出水，两个水壶之间，水也可以相互倾灌。已知 5 升壶为满壶，2 升壶为空壶，问如何通过倒水或灌水操作，使能在 2 升的壶中量出一升的水来。

1.3 对梵塔问题给出产生式系统描述，并讨论 N 为任意时状态空间的规模。

相传古代某处一庙宇中，有三根立柱，柱子上可套放直径不等的 N 个圆盘，开始时所有圆盘都放在第一根柱子上，且小盘处在大盘之上，即从下向上直径是递减的。和尚们的任务是把

所有圆盘一次一个地搬到另一个柱子上去（不许暂搁地上等），且小盘只许在大盘之上。问和尚们如何搬法最后能完成将所有的盘子都摞到第三根柱子上（其余两根柱子，有一根可作过渡盘子使用）。

求 $N=2$ 时，求解该问题的产生式系统描述，给出其状态空间图。讨论 N 为任意时，状态空间的规模。

1.4 对猴子摘香蕉问题，给出产生式系统描述。

一个房间里，天花板上挂有一串香蕉，有一只猴子可在房间里任意活动（到处走动，推移箱子，攀登箱子等）。设房间里还有一只可被猴子移动的箱子，且猴子登上箱子时才能摘到香蕉，问猴子在某一状态下（设猴子位置为 a ，箱子位置为 b ，香蕉位置为 c ），如何行动可摘取到香蕉。

1.5 对三枚钱币问题给出产生式系统描述及状态空间图。

设有三枚钱币，其排列处在“正、正、反”状态，现允许每次可翻动其中任意一个钱币，问只许操作三次的情况下，如何翻动钱币使其变成“正、正、正”或“反、反、反”状态。

1.6 说明怎样才能用一个产生式系统把十进制数转换为二进制数，并通过转换 141.125 这个数为二进制数，阐明其运行过程。

1.7 设可交换产生式系统的一条规则 R 可应用于综合数据库 D 来生成出 D' ，试证明若 R 存在逆，则可应用于 D' 的规则集等同于可应用于 D 的规则集。

1.8 一个产生式系统是以整数的集合作为综合数据库，新的数据库可通过把其中任意一对元素的乘积添加到原数据库的操作来产生。设以某一个整数子集的出现作为目标条件，试说明该产生式系统是可交换的。

1.9 讨论如何用产生式系统求解不定积分 $\int x^2 + 3x + \sin^2 x \cos^2 x dx$ 。

第二章 产生式系统的搜索策略

上一章讨论用产生式系统求解问题的过程中，已涉及到几种搜索策略的基本思想，当所给定的问题用状态空间表示时，则求解过程可归结为对状态空间的搜索。从图2.1表示出的搜索过程可以看出，当问题有解时，使用不同的搜索策略，找到解的搜索空间范围是有区别的。一般来说，对大空间问题，搜索策略是要解决组合爆炸的问题。

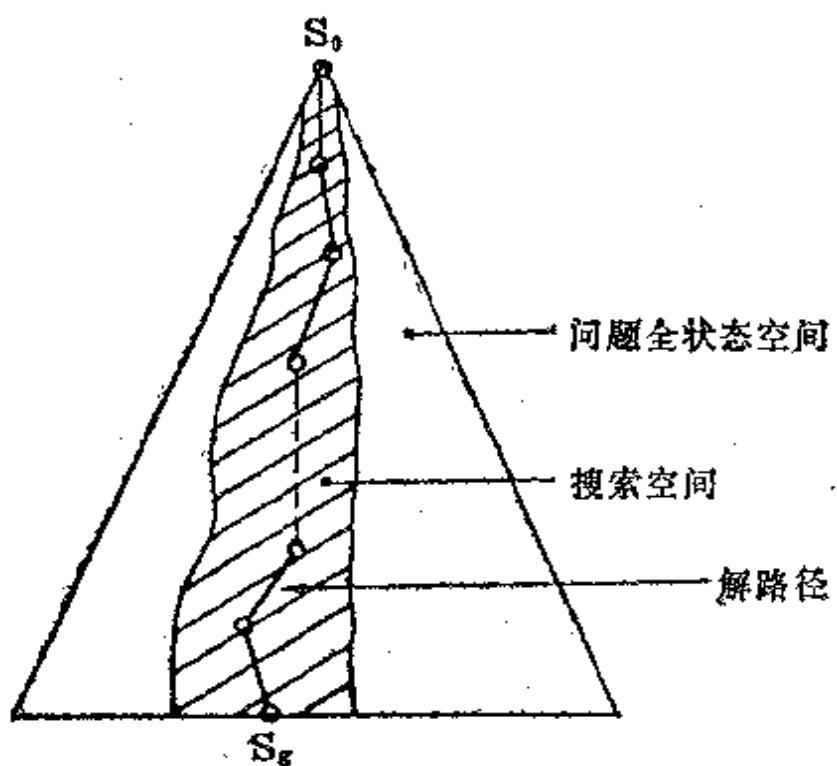


图 2.1 搜索空间示意图

通常搜索策略的主要任务是确定如何选取规则的方式。有两种基本方式：一种是不考虑给定问题所具有的特定知识，系统根据事先确定好的某种固定排序，依次调用规则或随机调用规则，

这实际上是盲目搜索的方法，一般统称为无信息引导的搜索策略。另一种是考虑问题领域可应用的知识，动态地确定规则的排序，优先调用较合适的规则使用，这就是通常称为启发式搜索策略或有信息引导的搜索策略。到目前为止，AI领域中已提出许多具体的搜索方法，概括起来有：

(1) 求任一解路的搜索策略

回溯法 (Backtracking)

爬山法 (Hill Climbing)

宽度优先法 (Breadth-first)

深度优先法 (Depth-first)

限定范围搜索法 (Beam Search)

好的优先法 (Best-first)

(2) 求最佳解路的搜索策略

大英博物馆法 (British Museum)

分枝界限法 (Branch and Bound)

动态规划法 (Dynamic Programming)

最佳图搜索法 (A*)

(3) 求与或关系解图的搜索法

一般与或图搜索法 (AO*)

极小极大法 (Minimax)

α - β 剪枝法 (Alpha-beta Pruning)

启发式剪枝法 (Heuristic Pruning)

本章及下一章仅对其中几个基本搜索策略作进一步的讨论。

2.1 回溯策略(Backtracking Strategies)

在1.3节中，我们对回溯控制策略作了一般讨论，并以八数码问题为例说明其用法。可以看出求解过程呈现出递归过程的性

质，因此用递归算法描述回溯控制下的产生式系统能抓住特点，简单有效。下面定义一个递归过程 BACKTRACK，它取单个自变量 DATA，设置为产生式系统的综合数据库，若算法成功结束，则返回一张规则表作为解输出；若找不到解，则算法返回 FAIL，失败退出。下面用类似 LISP 语言的形式给出一个具有两个回溯点（即设置两个回溯条件）的简单算法，并通过四皇后问题说明算法的运行过程。

1. 递归过程 BACKTRACK (DATA)

递归过程BACKTRACK (DATA)

- ① IF TERM (DATA) , RETURN NIL; TERM取真即找到目标，则过程返回空表 NIL。
- ② IF DEADEND (DATA), RETURN FAIL; DEAD-END取真，即该状态不合法，则过程返回 FAIL，必须回溯。
- ③ RULES:=APPRULES (DATA) ; APPRULES 计算 DATA 的可应用规则集，依某种原则（任意排列或按启发式准则）排列后赋给 RULES。
- ④ LOOP: IF NULL (RULES) , RETURN FAIL; NULL取真，即规则用完未找到目标，过程返回 FAIL，必须回溯。
- ⑤ R:=FIRST (RULES) ; 取头条可应用规则。
- ⑥ RULES:=TAIL (RULES) ; 删去头条规则，减少可应用规则表的长度。
- ⑦ RDATA:=GEN (R, DATA) ; 调用规则R作用于当前状态，生成新状态。
- ⑧ PATH:=BACKTRACK (RDATA) ; 对新状态递归调用本过程。
- ⑨ IF PATH=FAIL, GO LOOP; 当 PATH=FAIL 时，

递归调用失败，则转移调用另一规则进行测试。

⑩ RETURN CONS (R, PATH)；过程返回解路径规则表（或局部解路径子表）。

下面对这个算法作几点说明。首先解释一下其中的变量、常量、谓词、函数等符号的意义。变量符号 DATA、RULES、R、RDATA、PATH 分别表示当前状态、规则集序列表、当前调用规则、新生成状态、当前解路径表。常量符号 NIL、FAIL、LOOP 分别表示空表、回溯点标记、循环 标号。当状态变量 DATA 满足结束条件时，TERM (DATA) 取真；当 DATA 不具备构成解路上的状态时，DEADEND (DATA) 取真；当规则集变量表 RULES 取空时，NULL 取真。函数 RETURN、APPRULES、FIRST、TAIL、GEN、GO、CONS 的作用是：RETURN 返回其自变量值；APPRULES 求可应用规则集；FIRST 和 TAIL 分别取表头和表尾；GEN 调用规则 R 生成新状态；GO 执行转移；CONS 构造新表，把其第一个自变量加到一个表（第二个自变量）的前头。BACKTRACK (RDATA) 表示调用递归过程作用于新自变量上。

这里再说明一下该过程的运行情况。当某一个状态 S_x 满足结束条件时，算法才在第 1 步结束并返回 NIL，此时 BACKTRACK (S_x) = NIL。失败退出发生在第 2、4 步，第 2 步是处理不合法状态的回溯点，而第 4 步是处理全部规则应用均失败时的回溯点。若处在递归调用过程期间失败，过程会回溯到上一层继续运行，而在最高层失败则整个过程宣告失败退出。构造成功结束时的规则表在第 10 步完成。算法第 3 步实现可应用规则的排序，可以是固定排序或根据启发信息排序。

这个简单的 BACKTRACK 过程只设置两个回溯点，可用于求解皇后排列这类性质的问题，下面以四皇后问题为例来说明算法的运行过程。

2. 四皇后问题

在一个 4×4 的国际象棋棋盘上，一次一个地摆布四枚皇后棋子，摆好后要满足每行、每列和对角线上只允许出现一枚棋子，即棋子间不许相互俘获。

图 2.2 给出棋盘的几个势态，其中 a, b 满足目标条件，c, d, e 为不合法状态，即不可能构成满足目标条件的中间势态。

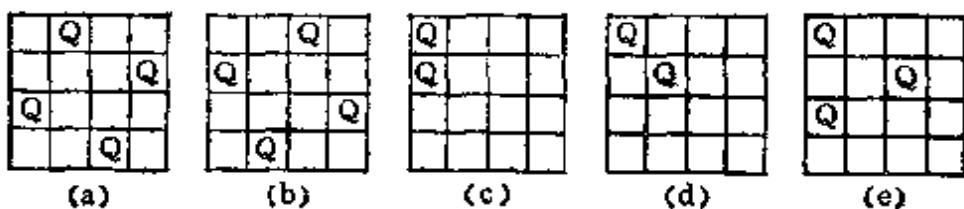


图 2.2 四皇后问题棋盘的几个势态

综合数据库：DATA = L (表)，L 的元素 $\subset \{ij\}$ ， $1 \leq i, j \leq 4$ 。DATA 非空时，其表元素表示棋子所在的行和列。因只有 4 个棋子，故表元素个数最多为 4。

图 2.2 中 a, b 分别记为 (12 24 31 43) 和 (13 21 34 42)，c, d, e 分别记为 (11 21)，(11 22)，(11 23 31) 等等。

规则集：if $1 \leq i \leq 4$ 且 Length DATA = $i - 1$

then APPEND (DATA (ij)) ($1 \leq j \leq 4$)

共 16 条规则，每条规则 R_{ij} 表示满足前提条件下，在 ij 处放一枚棋子。

当规则序列以 $R_{11}, R_{12}, R_{13}, R_{14}, R_{21}, \dots, R_{44}$ 这种固定排序方式调用 BACKTRACK 时，四皇后问题的搜索示意图如图 2.3 所示（为简单起见，每个状态只写出其增量部分）。可以看出，总共回溯 22 次，主过程结束时返回规则表 $(R_{12} R_{24} R_{31} R_{43})$ 。

对四皇后问题，如果要利用问题有关信息对规则进行动态排

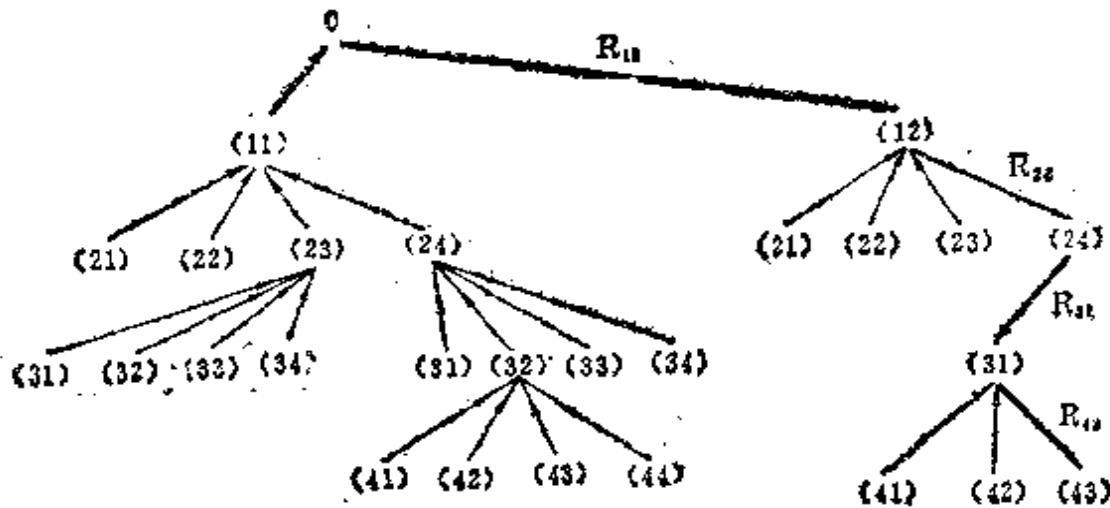


图 2.3 固定排序搜索树

序，则可定义一个对角线函数 $\text{diag}(i, j)$ 。该函数可计算出过棋盘上 ij 单元的最长的对角线长度，通过比较不同单元的 $\text{diag}(i, j)$ 函数值来决定 R_{ij} 的排序。如 $\text{diag}(i, k) < \text{diag}(i, j)$ ，则为 (R_{ik}, R_{ij}) ，即对角线短的单元，相应的规则排在前面；若 $\text{diag}(i, k) = \text{diag}(i, j)$ ，则仍为 (R_{ij}, R_{ik}) 。由此可计算得规则序列为 $R_{12} R_{13} R_{11} R_{14} \dots (0)$
 $R_{21} R_{24} R_{22} R_{23} R_{31} R_{34} R_{32} R_{33} R_{42}$
 $R_{43} R_{41} R_{44}$ ，这样所得搜索示意图如图 2.4 所示，只回溯 2 次就找到目标。

3. 递归过程 BACKTRACK1 (DATA LIST)

从四皇后的例子看出搜索深度有限，仅有 4 层，而且不可能出现重复状态的问题，因此 BACKTRACK 过程完全适用。对于八数码问题则不然，必须设置深度范

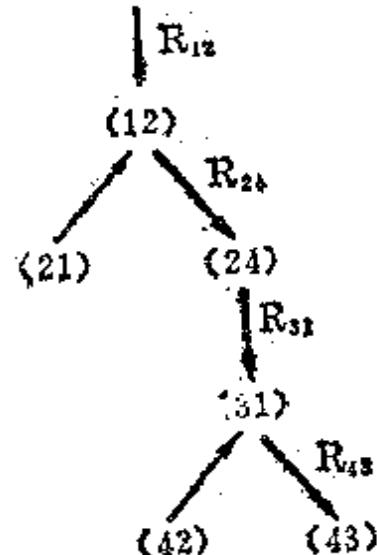


图 2.4 动态排序搜索树

围限制及防止出现重复状态引起的死循环这两个回溯点。改进后的算法如下：

递归过程 BACKTRACK1 (DATALIST)

- ① DATA:=FIRST (DATALIST) ,
- ② IF MEMBER (DATA TAIL (DATALIST)) , RETURN FAIL;
- ③ IF TERM (DATA) , RETURN NIL,
- ④ IF DEADEND (DATA) , RETURN FAIL,
- ⑤ IF LENGTH (DATALIST) > BOUND, RETURN FAIL;
- ⑥ RULES:=APPRULES (DATA) ,
- ⑦ LOOP: IF NULL (RULES) , RETURN FAIL;
- ⑧ R:=FIRST (RULES) ,
- ⑨ RULES:=TAIL (RULES) ,
- ⑩ RDATA:=GEN (R, DATA) ,
- ⑪ RDATALIST:=CONS (RDATA, DATALIST) ,
- ⑫ PATH:=BACKTRACK1 (RDATALIST) ,
- ⑬ IF PATH=FAIL, GO LOOP;
- ⑭ RETURN CONS (R, PATH) ,

这个算法与前者的差别是递归过程自变量是状态的链表，取返回到初始状态路径上所有状态组成的一张表，而 DATA 则是当前的一个状态。此外在第 2、5 步增设了两个回溯点以检验是否重访已出现过的状态和限定搜索深度范围，这分别由谓词 MEMBER 和 $>$ ，函数 LENGTH，常量 BOUND 实现。推广的回溯算法可应用于一般问题的求解，但这两个算法只描述了回溯一层的情况，即第 n 层递归调用失败，则控制退回到 (n-1) 层继续搜索。实际上往往造成深层搜索失败在于浅层原因引起，因此也可以利用启发信息，分析失败的原因，再回溯到合适的层次

上，这就是多层回溯策略的思想，目前已有一些系统使用了这种策略。

2.2 图搜索策略

产生式系统求解问题时，如果控制系统保留住所有规则应用后生成并链接起来的数据（状态）记录图，则称工作在这种方式下的控制系统使用了图搜索策略。实际上图搜索策略是实现从一个隐含图中，生成出一部分确实含有一个目标节点的显式表示子图的搜索过程。图搜索策略在 AI 系统中广泛被使用，本节将对算法及涉及的基本理论问题作进一步讨论。

首先回顾一下图论中几个术语的含义。

节点深度：根节点的深度为 0，其他节点的深度规定为父节点深度加 1，即 $d_{n+1} = d_n + 1$ 。

路径：设一节点序列为 $(n_0, n_1, \dots, n_i, \dots, n_k)$ ，对 $i=1, 2, \dots, k$ ，若节点 n_{i-1} 都具有一个后继节点 n_i ，则该节点序列为从节点 n_0 到节点 n_k 长度为 k 的一条路径。

路径耗散值：令 $C(n_i, n_j)$ 为节点 n_i 到 n_j 这段路径（或弧线）的耗散值，一条路径的耗散值等于连接这条路径各节点间所有弧线耗散值的总和。路径耗散值可按如下递归公式计算：

$$C(n_i, t) = C(n_i, n_j) + C(n_j, t)$$

$C(n_i, t)$ 为 $n_i \rightarrow t$ 这条路径的耗散值。

扩展一个节点：后继节点操作符（相当于可应用规则）作用到节点（对应于某一状态描述）上，生成出其所有后继节点（新状态），并给出连接弧线的耗散值（相当于使用规则的代价），这个过程叫做扩展一个节点。扩展节点可使定义的隐含图生成为显式表示的状态空间图。

一般图搜索算法

① $G := G_0$ ($G_0 = s$) , OPEN: = (s) , G 表示图, s 为初始节点, 设置 OPEN 表, 最初只含初始节点。

② CLOSED: = () , 设置 CLOSED 表, 起始设置为空表。

③ LOOP: IF OPEN = () , THEN EXIT (FAIL);

④ $n := \text{FIRST} (\text{OPEN})$, REMOVE(n , OPEN) , ADD (n , CLOSED) ; 称 n 为当前节点。

⑤ IF GOAL (n) , THEN EXIT (SUCCESS) ; 由 n 返回到 s 路径上的指针, 可给出解路径。

⑥ EXPAND (n) $\rightarrow \{m_i\}$, $G := \text{ADD } (m_i, G)$; 子节点集 $\{m_i\}$ 中不包含 n 的先辈节点。

⑦ 标记和修改指针

• ADD (m_i , OPEN) , 并标记 m_i 连接到 n 的指针; m_i 为 OPEN 和 CLOSED 中未出现过的子节点 $\{m_i\} = \{m_i\} \cup \{m_k\} \cup \{m_l\}$ 。

• 计算是否要修改 m_k , m_l 到 n 的指针; m_k 为已出现在 OPEN 中的子节点, m_l 为在 CLOSED 中的子节点。

• 计算是否要修改 m_i 到其后继节点的指针;

⑧ 对 OPEN 中的节点按某种原则重新排序;

⑨ GO LOOP;

这是一个很一般的图搜索过程, 通过不断的循环, 过程便生成出一个显式表示的图 G (搜索图) 和一个 G 的子集 T (搜索树)。该搜索树 T 是由第 7 步中标记的指针决定, 除根节点 s 外, G 中每个节点只有一个指针指向 G 中的一个父节点, 显然树中的每一个节点都处在 G 中。由于图 G 是无环的, 因此可根据树定义任一条特殊的路径。可以看出, OPEN 表上的节点, 都是搜索树的端节点, 即至今尚未被选作为扩展的节点, 而 CLOSED 表上的节点, 可以是已被扩展而不能生成后继节点的那些端节

点，也可以是树中的非端节点。

这个过程是在第 8 步要对 OPEN 表上的节点进行排序，以便在第 4 步能选出一个“最好”的节点优先扩展。不同的排序方法便可构成形式多样的专门搜索算法，这在后面还要进一步讨论。如果选出待扩的节点是目标节点，则算法在第 5 步成功结束，并可根据回溯到 s 的指针给出解路径。如果某个循环中，搜索树不再剩有待选的节点，即 OPEN 表变空时，则过程失败结束，问题找不到解。

现在再说明一下第 7 步中标记和修改指针的问题。如果要搜索的隐含图是一棵树，则可肯定第 6 步生成的后继节点不可能是以前生成过的，这时搜索图就是搜索树，不存在 m_k, m_l 这种类型的子节点，因此不必进行修改指针的操作。如果要搜索的隐含图不是一棵树，则有可能出现 m_k 这样的子节点，就是说这时又发现了到达 m_k 的新通路，这样就要比较不同路径的耗散值，把指针修改到具有较小耗散值的路径上。例如，图 2.5 所示的两个

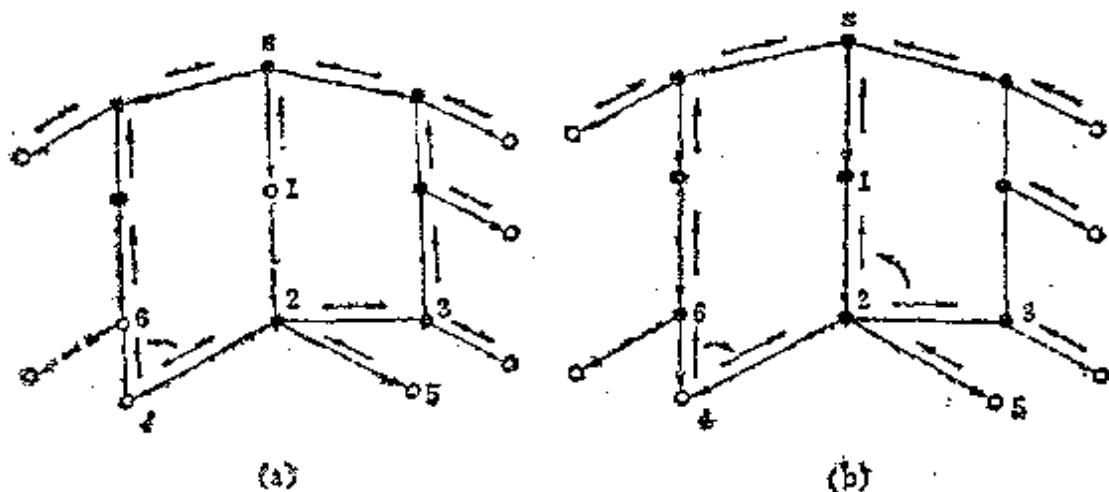


图 2.5 扩展节点 6 和节点 1 得到的搜索图

搜索图中，实心圆点在 CLOSED 表中（已扩展过的节点），空心圆点则在 OPEN 表中（待扩展点）。先设下一步轮到要扩展节点 6，并设生成的两个子节点，其中有一个 4 已在 OPEN 中，

那么原先路径 $(s \rightarrow 3 \rightarrow 2 \rightarrow 4)$ 耗散值为 5 (设每段路径均为单位耗散)，新路径 $(s \rightarrow 6 \rightarrow 4)$ 的耗散值为 4，所以 4 的指针应由指向 2 修正到指向 6，如图 2.5 (a) 所示。接着设下一循环要扩展节点 1，若节点 1 只生成一个子节点 2 (它已在 CLOSED 上)，显然这时节点 2 原先指向节点 3 的指针，要修改到指向节点 1，由此又引起子节点 4 的指针又修改为指向 2，如图 2.5 (b) 所示。

2.3 无信息图搜索过程

无信息图搜索过程是在算法的第 8 步中使用任意排列 OPEN 表节点的顺序，通常有两种排列方式：

1. 深度优先

过程 DEPTH-FIRST-SEARCH

- ① $G := G_0$ ($G_0 = s$) , OPEN:=(s), CLOSED:=() ,
- ② LOOP: IF OPEN = () THEN EXIT (FAIL) ,
- ③ $n := \text{FIRST}(\text{OPEN})$,
- ④ IF GOAL (n) THEN EXIT (SUCCESS) ,
- ⑤ REMOVE (n , OPEN), ADD (n , CLOSED) ,
- ⑥ EXPAND (n) $\rightarrow \{m_i\}$,
 $G := \text{ADD} (m_i, G)$,
- ⑦ ADD (m_i , OPEN) , 并标记 m_i 到 n 的指针；把不在 OPEN 或 CLOSED 中的节点放到 OPEN 表的最前面，使深度大的节点可优先扩展。
- ⑧ GO LOOP;

2. 宽度优先

过程 BREADTH-FIRST-SEARCH

```

① G:=G0 (G0 ⇒ s) , OPEN:=(s) , CLOSED:=( ) ;
② LOOP: IF OPEN=( ) THEN EXIT (FAIL) ;
③ n:=FIRST (OPEN) ;
④ IF GOAL (n) THEN EXIT (SUCCESS) ;
⑤ REMOVE (n, OPEN) , ADD (n, CLOSED) ;
⑥ EXPAND (n) → {mi} ,
   G:=ADD (mi, G) ,
⑦ ADD (OPEN, mi) , 并标记 mi 到 n 的指针；把不在
OPEN 或 CLOSED 中的节点放在 OPEN 表的后面，使深度浅的
节点可优先扩展。
⑧ GO LOOP;

```

一般情况下，使用深度优先法要对搜索深度事先给出某种限制。当问题有解时，这两种方法都能保证找到解，在单位耗散的条件下，宽度优先法还能保证找到最短路径。此外对复杂 NP 完全类问题，一般不可避免会产生指数爆炸。

2.4 启发式图搜索过程

启发式搜索是利用问题拥有的启发信息来引导搜索，达到减少搜索范围，降低问题复杂度的目的。这种利用启发信息的搜索过程都称为启发式搜索方法。在研究启发式搜索方法时，先说明一下启发信息应用，启发能力度量及如何获得启发信息这几个问题，然后再来讨论算法及一些理论问题。

一般来说启发信息过弱，产生式系统在找到一条路径之前将扩展过多的节点，即求得解路径所需搜索的耗散值（搜索花费的工作量）较大；相反引入强的启发信息，有可能大大降低搜索工作量，但不能保证找到最小耗散值的解路径（最佳路径），因此实际应用中，希望最好能引入降低搜索工作量的启发信息而不牺

牲找到最佳路径的保证。这是一个重要而又困难的问题，从理论上要研究启发信息和最佳路径的关系，从实际上则要解决获取启发信息方法的问题。

比较不同搜索方法的效果可用启发能力的强弱来度量。在大多数实际问题中，人们感兴趣的是使路径的耗散值和求得路径所需搜索的耗散值两者的某种组合最小，更一般的情况是考虑搜索方法对求解所有可能遇见的问题，其平均的组合耗散值最小。如果搜索方法 1 的平均组合耗散值比方法 2 的平均组合耗散值低，则认为方法 1 比方法 2 有更强的启发能力。实际上很难给出一个计算平均组合耗散值的方法，因此启发能力的度量也只能根据使用方法的实际经验作出判断，没有必要去追求严格地比较结果。

启发式搜索过程中，要对 OPEN 表进行排序，这就需要有一种方法来计算待扩展节点有希望通向目标节点的不同程度，我们总是希望能找到最有希望通向目标节点的待扩展节点优先扩展。一种最常用的方法是定义一个评价函数 f (Evaluation function) 对各个节点进行计算，其目的就是用来估算出“有希望”的节点来。如何定义一个评价函数呢？通常可以参考的原则有：一个节点处在最佳路径上的概率；求出任意一个节点与目标节点集之间的距离度量或差异度量；根据格局（博弈问题）或状态的特点来打分。即根据问题的启发信息，从概率角度、差异角度或记分法给出计算评价函数的方法。

1. 启发式搜索算法 A

利用评价函数 $f(n) = g(n) + h(n)$ 来排列 OPEN 表节点顺序的图搜索算法称为算法 A。

过程 A

- ① OPEN:=(s), $f(s) := g(s) + h(s)$,
- ② LOOP: IF OPEN=() THEN EXIT(FAIL),

③ $n := \text{FIRST}(\text{OPEN})$;
④ IF GOAL (n) THEN EXIT (SUCCESS) ;
⑤ REMOVE (n , OPEN) , ADD (n , CLOSED);
⑥ EXPAND (n) $\rightarrow \{m_i\}$, 计算 $f(n, m_i) = g(n, m_i) + h(m_i)$; $g(n, m_i)$ 是从 s 通过 n 到 m_i 的耗散值, $f(n, m_i)$ 是从 s 通过 n 、 m_i 到目标节点耗散值的估计。

• ADD (m_i , OPEN) , 标记 m_i 到 n 的指针;
• IF $f(n, m_k) < f(m_k)$ THEN $f(m_k) := f(n, m_k)$,
标记 m_k 到 n 的指针, 比较 $f(n, m_k)$ 和 $f(m_k)$, $f(m_k)$ 是
扩展 n 之前计算的耗散值。
• IF $f(n, m_i) < f(m_i)$ THEN $f(m_i) := f(n, m_i)$,
标记 m_i 到 n 的指针, ADD (m_i , OPEN) ; 把 m_i 重放回 OPEN
中, 不必考虑修改到其子节点的指针。

⑦ OPEN 中的节点按 f 值从小到大排序;
⑧ GO LOOP;

算法中 $f(n)$ 规定为对从初始节点 s 出发, 约束通过节点 n 到达目标节点 t 上, 最小耗散值路径的耗散值 $f^*(n)$ 的估计值, 通常取正值。 $f(n)$ 由两个分量组成, 其中 $g(n)$ 是到目前为止, 从 s 到 n 的一条最小耗散值路径的耗散值, 是作为从 s 到 n 最小耗散值路径的耗散值 $g^*(n)$ 的估计值, $h(n)$ 是从 n 到目标节点 t 上, 最小耗散值路径的耗散值 $h^*(n)$ 的估计值。

设函数 $k(n, n_i)$ 表示最小耗散路径的实际耗散值 (当 n_i 到 n_i 无通路时则 $k(n_i, n_i)$ 无意义), 则 $g^*(n) = k(s, n)$, $h^*(n) = \min k(n, t_i)$, 其中 t_i 是目标节点集, $k(n, t_i)$ 就是从 n 到每一个目标节点最小耗散值路径的耗散值, $h^*(n)$ 是其中最小值的那条路径的耗散值, 而具有 $h^*(n)$ 值的路径是 n 到 t_i 的最佳路径。由此可得 $f^*(n) = g^*(n) + h^*(n)$ 就表示 $s \rightarrow t_i$ 并约束通过节点 n 的最佳路径的耗散值。当 $n = s$ 时, $f^*(s)$

$= h^*(s)$ 则表示 $s \rightarrow t_1$ 无约束的最佳路径的耗散值, 这样一来, 所定义的 $f(n) = g(n) + h(n)$ 就是对 $f^*(n)$ 的一个估计。 $g(n)$ 的值实际上很容易从到目前为止的搜索树上计算出来, 不必专门定义计算公式, 也就是根据搜索历史情况对 $g^*(n)$ 作出估计, 显然有 $g(n) \geq g^*(n)$ 。 $h(n)$ 则依赖于启发信息, 通常称为启发函数, 是要对未来扩展的方向作出估计。算法 A 是按 $f(n)$ 递增的顺序来排列 OPEN 表的节点, 因而优先扩展 $f(n)$ 值小的节点, 体现了好的优先搜索思想, 所以算法 A 是一个好的优先的搜索策略。图 2.6 表示出当前要扩展节点 n 之前的搜索

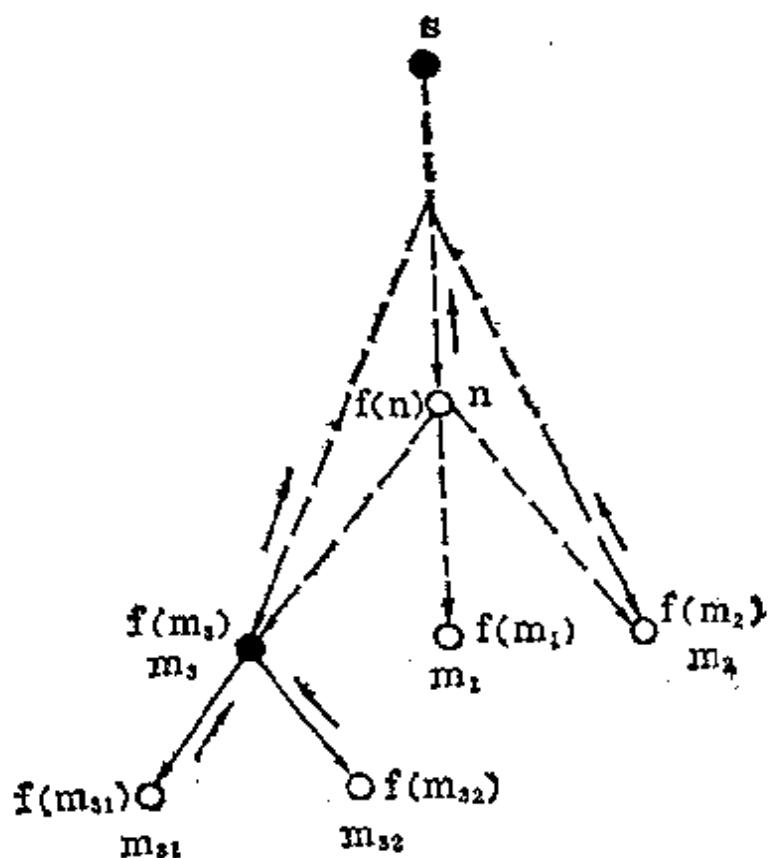


图 2.6 搜索示意图

图, 扩展 n 后新生成的子节点 $m_1 (\in \{m_j\})$ 、 $m_2 (\in \{m_k\})$ 、 $m_3 (\in \{m_l\})$ 要分别计算其评价函数值:

$$f(m_1) = g(m_1) + h(m_1)$$

$$f(n, m_2) = g(n, m_2) + h(m_2)$$

$$f(n, m_3) = g(n, m_3) + h(m_3)$$

然后按第 6 步条件进行指针设置和第 7 步重排 OPEN 表节点顺序，以便确定下一次要扩展的节点。

下面再以八数码问题为例说明好的优先搜索策略的应用过程。设评价函数 $f(n)$ 形式如下：

$$f(n) = d(n) + W(n)$$

其中 $d(n)$ 代表节点的深度，取 $g(n) = d(n)$ 表示讨论单位耗散的情况；取 $h(n) = W(n)$ 表示以“不在位”的将牌个数作为启发函数的度量，这时 $f(n)$ 可估计出通向目标节点的希望程度。图 2.7 表示使用这种评价函数时的搜索树，图中括弧中的数字表示该节点的评价函数值。算法每一循环结束时，其 OPEN 表和 CLOSED 表的排列如下：

OPEN表	CLOSED表
初始化 $(s(4))$	$()$
第一循环结束 $(B(4) A(6) C(6))$	$(s(4))$
第二循环结束 $(D(5) E(5) A(6) C(6) F(6))$	$(s(4) B(4))$
第三循环结束 $(E(5) A(6) C(6) F(6) G(6) H(7))$	$(s(4) B(4) D(5))$
第四循环结束 $(I(5) A(6) C(6) F(6) G(6) H(7) J(7))$	$(s(4) B(4) D(5) E(5))$
第五循环结束 $(K(5) A(6) C(6) F(6) G(6) H(7) J(7))$	$(s(4) B(4) D(5) E(5) I(5))$
第六循环结束 $(L(5) A(6) C(6) F(6) G(6) H(7) J(7) M(7))$	$(s(4) B(4) D(5) E(5) I(5) K(5))$
第七循环结束 第四步成功退出	

根据目标节点 L 返回到 s 的指针，可得解路径

S(4), B(5), E(5), I(5), K(5), L(5)

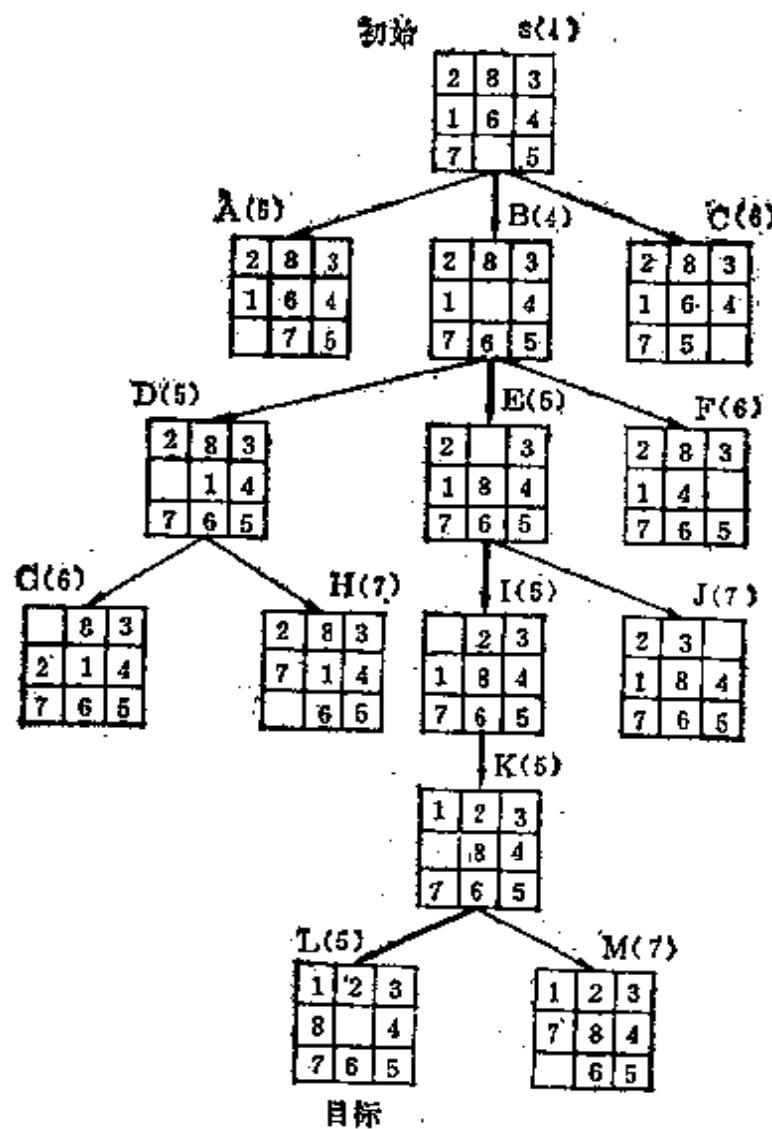


图 2.7 八数码问题的搜索树

2. 爬山法

过程 Hill-climbing

① $n := s$; s 为初始节点

- ② LOOP: IF GOAL(n) THEN EXIT (SUCCESS) ;
- ③ EXPAND (n) $\rightarrow \{m_i\}$, 计算 $h(m_i)$, $nextn := m_i$ (min $h(m_i)$ 的节点) ;
- ④ IF $h(n) < h(nextn)$ THEN EXIT(FAIL) ;
- ⑤ $n := nextn$;
- ⑥ GO LOOP;

显然如果将山顶作为目标, $h(n)$ 表示山顶与当前位置 n 之间高度之差, 则该算法相当于总是登上山顶, 在单峰的条件下, 必能到达山峰。

3. 分支界限法

分支界限法是优先扩展当前具有最小耗散值分支路径的端节点 n , 其评价函数为 $f(n) = g(n)$ 。该法的基本思想是简单的, 实际上是建立一个局部路径(或分支)的队列表, 每次都选耗散值最小的那个分支上的端节点来扩展, 直到生成出含有目标节点的路径为止。

过程 Branch-Bound

- ① QUEUE:=($s - s$), $g(s) = 0$;
- ② LOOP: IF QUEUE=() THEN EXIT (FAIL);
- ③ PATH:=FIRST(QUEUE), $n := LAST(PATH)$;
- ④ IF GOAL(n) THEN EXIT(SUCCESS);
- ⑤ EXPAND(n) $\rightarrow \{m_i\}$, 计算 $g(m_i) = g(n, m_i)$,
REMOVE($s - n$, QUEUE), ADD($s - m_i$, QUEUE);
- ⑥ QUEUE 中局部路径 g 值最小者排在前面;
- ⑦ GO LOOP;

应用该算法求解图 2.8 的最短路径问题, 其搜索图如图 2.9 所示, 求解过程中 QUEUE 的结果简记如下(D(4)代表耗散值为 4 的 s -D 分支, 其余类推) :

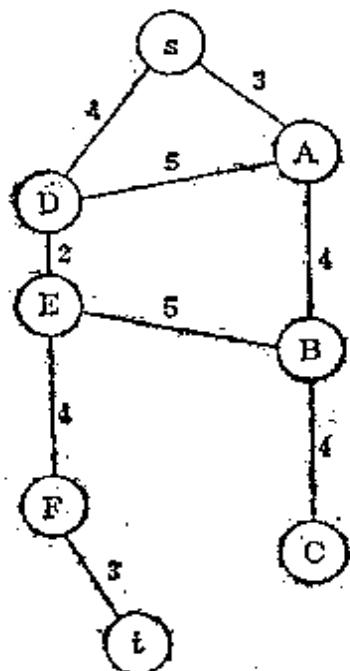


图 2.8 八城市地图网示意图

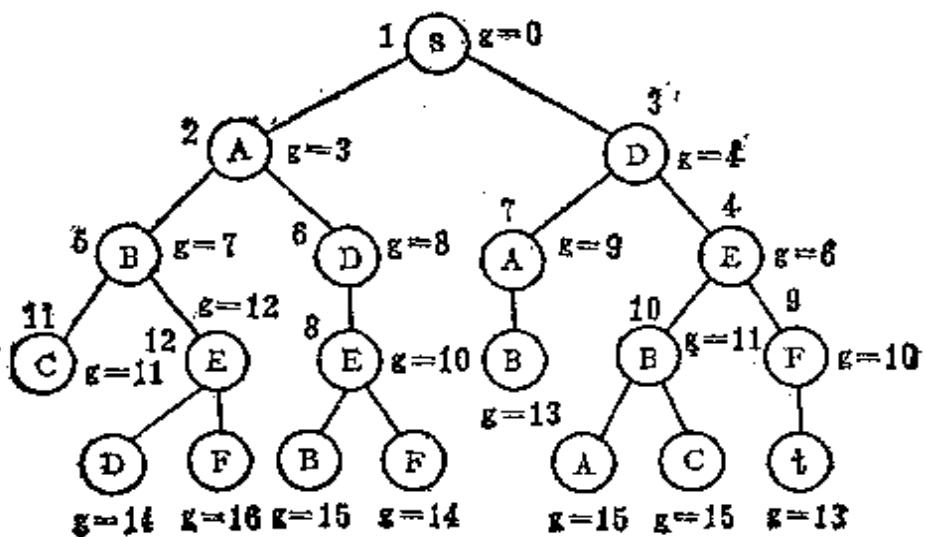


图 2.9 分支界限搜索树

初始 $(s(0))$

- 1) $(A(3) \ D(4))$
- 2) $(D(4) \ B(7) \ D(8))$

- 3) (E(6) B(7) D(8) A(9))
- 4) (B(7) D(8) A(9) F(10) B(11))
- 5) (D(8) A(9) F(10) B(11) C(11) E(12))
- 6) (A(9) E(10) F(10) B(11) C(11) E(12))
- 7) (E(10) F(10) B(11) C(11) E(12) B(13))
- 8) (F(10) B(11) C(11) E(12) B(13) F(14) B(15))
- 9) (B(11) C(11) E(12) t(13) B(13) F(14) B(15))
- 10) (C(11) E(12) t(13) B(13) F(14) A(15) B(15) C(15))
- 11) (E(12) t(13) B(13) F(14) A(15) B(15) C(15))
- 12) (t(13) B(13) F(14) D(14) A(15) B(15) C(15) F(16))
- 13) 结束。

4. 动态规划法

动态规划法实际上是对分支界限法的改进。从图2.9看出,第二循环扩展 A(3) 后生成的 D(8) 节点(D(4) 已在QUEUE上)和第三循环扩展D(4)之后生成的A(9)节点(A(3)已在QUEUE上)都是多余的分支,因为由s—D到达目标的路径显然要比s—A—D到达目标的路径要好。因此删去类似于s—A—D或s—D—A这样一些多余的路径将会大大提高搜索效率。动态规划原理指出,求 $s \rightarrow t$ 的最佳路径时,对某一个中间节点 I, 只要考虑 s 到 I 中具有最小耗散值这一条局部路径就可以,其余 s 到 I 的路径是多余的,不必加以考虑。下面给出具有动态规划原理的分支界限算法。

过程 Dynamic-Programming

- ① QUEUE:=(s-s), $g(s)=0$;
- ② LOOP: IF QUEUE: () THEN EXIT(FAIL);

- ③ PATH:=FIRST(QUEUE), n:=LAST(PATH);
- ④ IF GOAL(n) THEN EXIT(SUCCESS);
- ⑤ EXPAND(n) $\rightarrow \{m_i\}$, 计算 $g(m_i) = g(n, m_i)$, REMOVE(s-n, QUEUE), ADD(s-m_i, QUEUE);
- ⑥ 若QUEUE中有多条到达某一公共节点的路径, 则只保留耗散值最小的那条路径, 其余删去, 并重新排序, g值最小者排在前面;

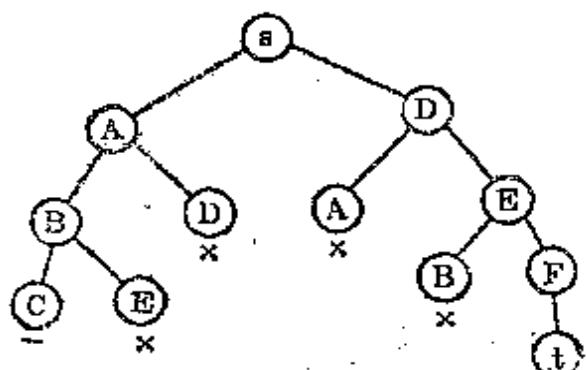


图 2.10 动态规划原理的搜索树

- ⑦ GO LOOP;

对图2.8的例子应用该算法, 其搜索图如图2.10所示, 实际只剩下两条搜索路径, 改善了搜索效率。

5. 最佳图搜索算法 A*(Optimal Search)

当在算法A的评价函数中, 使用的启发函数 $h(n)$ 是处在 $h^*(n)$ 的下界范围, 即满足 $h(n) \leq h^*(n)$ 时, 则我们把这个算法称为算法A*。A*算法实际上是分支界限和动态规划原理及使用下界范围的h函数相结合的算法。当问题有解时, A*一定能找到一条到达目标节点的最佳路径。例如在极端情况下, 若 $h(n) \equiv 0$ (肯定满足下界范围条件) 因而一定能找到最佳路径。此时若 $g=d$, 则算法等同于宽度优先算法。前面已提到过, 宽度优先算法能保证找到一条到达目标节点最小长度的路径, 因而这个特例从直观上就验证了A*的一般结论。

一般地说对任意一个图, 当s到目标节点有一条路径存在时, 如果搜索算法总是在找到一条从s到目标节点的最佳路径上结束, 则称该搜索算法是可采纳的(Admissibility)。A*就具有可采纳性, 下面就来证明A*的可采纳性及若干重要性质。

定理 1：对有限图，如果从初始节点 s 到目标节点 t 有路径存在，则算法 A 一定成功结束。

证明：设 A 搜索失败，则算法在第 2 步结束，OPEN 表变空，而 CLOSED 表中的节点是在结束之前被扩展过的节点。由于图有解，令 $(n_0 = s, n_1, n_2, \dots, n_k = t)$ 表示某一任一解路，我们从 n_k 开始逆向逐个检查该序列的节点，找到出现在 CLOSED 表中的节点 n_i ，即 $n_i \in \text{CLOSED}, n_{i+1} \notin \text{CLOSED}$ (n_i 一定能找到，因为 $n_0 \in \text{CLOSED}, n_k \notin \text{CLOSED}$)。由于 n_i 在 CLOSED 中，必定在第 6 步被扩展，且 n_{i+1} 被加到 OPEN 中，因此在 OPEN 表空之前， n_{i+1} 已被处理过。若 n_{i+1} 是目标节点，则搜索成功，否则它被加入到 CLOSED 中，这两种情况都与搜索失败的假设矛盾，因此对有限图不失败则成功。〔证毕〕

因为 A^* 是 A 的特例，因此它具有 A 的所有性质。这样对有限图如果有解，则 A^* 一定能在找到到达目标的路径结束，下面要证明即使是无限图， A^* 也能找到最佳解结束。我们先证两个引理：

引理 2.1：对无限图，若有从初始节点 s 到目标节点 t 的一条路径，则 A^* 不结束时，在 OPEN 中即使最小的一个 f 值也将增到任意大，或有 $f(n) > f^*(s)$ 。

证明：设 $d^*(n)$ 是 A^* 生成的搜索树中，从 s 到任一节点 n 最短路径长度的值（设每个弧的长度均为 1），搜索图上每个弧的耗散值为 $C(n_i, n_{i+1})$ (C 取正)。令 $e = \min C(n_i, n_{i+1})$ ，则 $g^*(n) \geq d^*(n)e$ 。而 $g(n) \geq g^*(n) \geq d^*(n)e$ ，故有

$f(n) = g(n) + h(n) \geq g(n) \geq d^*(n)e$ (设 $h(n) \geq 0$)。若 A^* 不结束， $d^*(n) \rightarrow \infty$ ， f 值将增到任意大。

设 $M = \frac{f^*(s)}{e}$ ， M 是一个定数，所以搜索进行到一定程度会

有 $d^*(n) > M$, 或 $\frac{d^*(n)}{M} > 1$, 则

$$f(n) \geq d^*(n) \Rightarrow d^*(n) \cdot \frac{f^*(s)}{M} = f^*(s) \cdot \frac{d^*(n)}{M} > f^*(s).$$

〔证毕〕

引理 2.2: A*结束前, OPEN 表中必存在 $f(n) \leq f^*(s)$ 的节点 (n 是在最佳路径上的节点)。

证明: 设从初始节点 s 到目标节点 t 的一条最佳路径序列为 $(n_0 = s, n_1, \dots, n_k = t)$

算法初始化时, s 在 OPEN 中, 由于 A* 没有结束, 在 OPEN 中存在最佳路径上的节点。设 OPEN 表中的第一个节点 n 是处在最佳路径序列中 (至少有一个这样的节点, 因 s 一开始是在 OPEN 上), 显然 n 的先辈节点 n_0 已在 CLOSED 中, 因此能找到 s 到 n_0 的最佳路径, 而 n 也在最佳路径上, 因而 s 到 n 的最佳路径也能找到, 因此有

$$\begin{aligned} f(n) &= g(n) + h(n) = g^*(n) + h(n) \\ &\leq g^*(n) + h^*(n) = f^*(n) \end{aligned}$$

而最佳路径上任一节点均有 $f^* = f^*(s)$ ($f^*(s)$ 是最佳路径的耗散值), 所以 $f(n) \leq f^*(s)$ 。〔证毕〕

定理 2: 对无限图, 若从初始节点 s 到目标节点 t 有路径存在, 则 A* 也一定成功结束。

证明: 假定 A* 不结束, 由引理 2.1 有 $f(n) > f^*(s)$, 或 OPEN 表中最小的一个 f 值也变成无界, 这与引理 2.2 的结论矛盾, 所以 A* 只能成功结束。〔证毕〕

推论 2.1: OPEN 表上任一具有 $f(n) < f^*(s)$ 的节点 n, 最终都将被 A* 选作为扩展的节点。

定理 3: 若存在初始节点 s 到目标节点 t 的路径, 则 A* 必

能找到最佳解结束。

证明：（1）由定理1、2知A*一定会找到一个目标节点结束。

（2）设找到的一个目标节点t结束，而 $s \rightarrow t$ 不是一条最佳路径，即

$$f(t) = g(t) > f^*(s)$$

而根据引理2.2知结束前OPEN表上有节点n，且处在最佳路径上，并有 $f(n) \leq f^*(s)$ ，所以

$$f(n) \leq f^*(s) < f(t)$$

这时算法A*应选n作为当前节点扩展，不可能选t，从而也不会去测试目标节点t，即这与假定A*选t结束矛盾，所以A*只能结束在最佳路径上。〔证毕〕

推论3.1：A*选作扩展的任一节点n，有 $f(n) \leq f^*(s)$ 。

证明：令n是由A*选作扩展的任一节点，因此n不会是目标节点，且搜索没有结束，由引理2.2而知在OPEN中有满足 $f(n') \leq f^*(s)$ 的节点n'。若 $n = n'$ ，则 $f(n) \leq f^*(s)$ ，否则选n扩展，必有 $f(n) \leq f(n')$ ，所以 $f(n) \leq f^*(s)$ 成立。〔证毕〕

6. 启发函数与A*算法的关系

应用A*的过程中，如果选作扩展的节点n，其评价函数值 $f(n) = f^*(n)$ ，则不会去扩展多余的节点就可找到解。可以想像到 $f(n)$ 越接近于 $f^*(n)$ ，扩展的节点数就会越少，即启发函数中，应用的启发信息（问题知识）愈多，扩展的节点数就较少。

定理4：有两个A*算法 A_1 和 A_2 ，若 A_2 比 A_1 有较多的启发信息（即对所有非目标节点均有 $h_2(n) > h_1(n)$ ），则在具有一条从s到t的隐含图上，搜索结束时，由 A_2 所扩展的每一个节点，也必定由 A_1 所扩展，即 A_1 扩展的节点数至少和 A_2 一样多。

证明：使用数学归纳法，对节点的深度应用归纳法。

(1) 对深度 $d(n)=0$ 的节点 (即初始节点 s)，定理结论成立，即若 s 为目标节点，则 A_1 和 A_2 都不扩展 s ，否则 A_1 和 A_2 都扩展了 s (归纳法前提)。

(2) 设深度 $d(n) \leq k$ 时，对所有路径的端节点，定理结论都成立 (归纳法假设)。

(3) 要证明 $d(n)=k+1$ 时，所有路径的端节点，结论成立，我们用反证法证明。

设 A_2 搜索树上有一个节点 n ($d(n)=k+1$) 被 A_2 扩展了，而对应于 A_1 搜索树上的这个节点 n ，没有被 A_1 扩展。根据归纳法假设条件， A_1 扩展了 n 的父节点， n 是在 A_1 搜索树上，因此 A_1 结束时， n 必定保留在其 OPEN 表上， n 没有被 A_1 选择扩展，有

$$f_1(n) \geq f^*(s), \text{ 即 } g_1(n) + h_1(n) \geq f^*(s)$$

所以 $h_1(n) \geq f^*(s) - g_1(n)$ (1)

另一方面 A_2 扩展了 n ，有

$$f_2(n) \leq f^*(s), \text{ 即 } g_2(n) + h_2(n) \leq f^*(s)$$

所以 $h_2(n) \leq f^*(s) - g_2(n)$ (2)

由于 $d=k$ 时， A_2 扩展的节点， A_1 也一定扩展，故有

$$g_1(n) \leq g_2(n) \quad (\text{因 } A_1 \text{ 扩展的节点数可能较多})$$

所以 $h_1(n) \geq f^*(s) - g_1(n) \geq f^*(s) - g_2(n)$ (3)

比较式 (2)、(3) 可得：至少在节点 n 上有 $h_1(n) \geq h_2(n)$ ，这与定理的前提条件矛盾，因此存在节点 n 的假设不成立。[证毕]

根据这个定理可知，使用启发函数 $h(n)$ 的 A^* 算法，比不使用 $h(n)$ ($h(n) \equiv 0$) 的算法，求得最佳路径时扩展的节点数要少，图 2.11 的搜索图例子可看出比较的结果。当 $h \equiv 0$ 时，求得最佳解路 (s, C, J, t_1) ，其 $f^*(s) = 8$ ，但除 $t_1 \sim t_8$ 外所有节点都扩展了，即求出所有解路后，才找到耗散值最小的路径。而引入启发函数 (设其函数值如图中节点旁边所示) 后，除了最佳路径上的节点 s, C, J 被扩展外，其余的节点都没有被扩展。当然

一般情况下，并不一定都能达到这种效果，主要在于获取完备的启发信息较为困难。

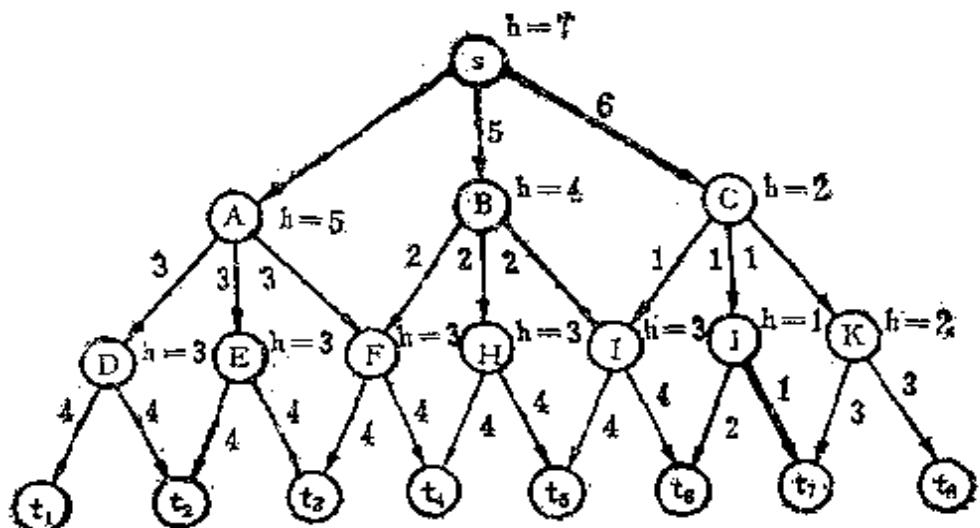


图 2.11 启发函数 $h(n)$ 的效果比较

7. A* 算法的改进

上一节讨论了启发函数对扩展节点数所起的作用。如果用扩展节点数作为评价搜索效率的准则，那么可以发现 A 算法第 6 步中，对 m_1 类节点要重新放回 OPEN 表中的操作，将引起多次扩展同一个节点的可能，因而即使扩展的节点数少，但重复扩展某些节点，也将导致搜索效率下降。图 2.12 给出同一节点多次扩展的例子，下页列出调用算法 A* 过程时 OPEN 和 CLOSED 表的状态。从 CLOSED 表看出，在修改 m_1 类节点指针过程中，节点 A，B，C 重复扩展，次数分别为 8、4、2，总共扩展 16 次节点。

从这个例子看出，如果不使用启发函数，则每个节点仅扩展一次，虽然扩展的节点数相同，但 A* 扩展的次数多。如果对启发函数施加一定的限制——单调限制，则当 A* 算法选某一个节点扩展时，就已经找到到该节点的最佳路径。下面就来证明这个

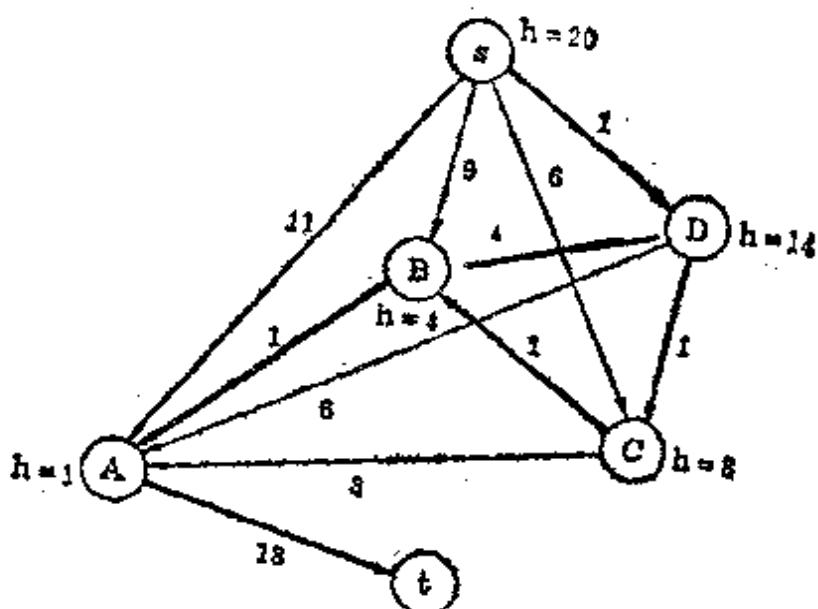


图 2.12 A*算法多次扩展同一节点搜索图例子

OPEN 表	CLOSED 表
初始化 $(s(20))$	$()$
1 $(A(12) B(13) C(14) D(15))$	$(s(20))$
2 $(B(13) C(14) D(15) t(29))$	$(s(20) A(12))$
3 $(A(11) C(14) D(15) t(29))$	$(s(20) B(13))$
4 $(C(14) D(15) t(28))$	$(s(20) A(11) B(13))$
5 $(A(10) B(11) D(15) t(28))$	$(s(20) C(14))$
6 $(B(11) D(15) t(27))$	$(s(20) A(10) C(14))$
7 $(A(9) D(15) t(27))$	$(s(20) B(11) C(14))$
8 $(D(15) t(26))$	$(s(20) A(9) B(11) C(14))$
9 $(A(8) B(9) C(10) t(26))$	$(s(20) D(15))$
10 $(B(9) C(10) t(25))$	$(s(20) A(8) D(15))$
11 $(A(7) C(10) t(25))$	$(s(20) B(9) D(15))$
12 $(C(10) t(24))$	$(s(20) A(7) B(9) D(15))$
13 $(A(6) B(7) t(24))$	$(s(20) C(10) D(15))$
14 $(B(7) t(23))$	$(s(20) A(6) C(10) D(15))$
15 $(A(5) t(23))$	$(s(20) B(7) C(10) D(15))$
16 $(t(22))$	$(s(20) A(5) B(7) C(10) D(15))$
17 成功结束	

结论。

一个启发函数 h , 如果对所有节点 n_i 和 n_j (n_i 是 n_j 的子节点), 都有 $h(n_i) - h(n_j) \leq C(n_i, n_j)$ 或 $h(n_i) \leq C(n_i, n_j) + h(n_j)$ 且 $h(n_i) = 0$, 则称该 h 函数满足单调限制条件。其意义是从 n_i 到目标节点, 最佳路径耗散值估计 $h(n_i)$ 不大于 n_i 到目标节点最佳路径耗散值估计 $h(n_j)$ 与 n_i 到 n_j 弧线耗散值两者之和。

对八数码问题, $h(n) = W(n)$ 是满足单调限制的条件。

定理 5: 若 $h(n)$ 满足单调限制条件, 则 A^* 扩展了节点 n 之后, 就已经找到了到达节点 n 的最佳路径。即若 A^* 选 n 来扩展, 在单调限制条件下有 $g(n) = g^*(n)$ 。

证明: 设 n 是 A^* 选作扩展的任一节点, 若 $n=s$, 显然有 $g(s) = g^*(s) = 0$, 因此考虑 $n \neq s$ 的情况。

我们用序列 $P = (n_0=s, n_1, \dots, n_k=n)$ 表达到达 n 的最佳路径。现在从 OPEN 中取出非初始节点 n 扩展时, 假定没有找到 P , 这时 CLOSED 中一定会有 P 中的节点 (至少 s 是在 CLOSED 中, n_k 刚被选作扩展, 不在 CLOSED 中), 把 P 序列中 (依顺序检查) 最后一个出现在 CLOSED 中的节点称为 n_i , 那么 n_{i+1} 是在 OPEN 中 ($n_{i+1} \neq n$), 由单调限制条件, 对任意 i 有

$$g^*(n_i) + h(n_i) \leq g^*(n_{i+1}) + C(n_i, n_{i+1}) + h(n_{i+1}) \quad (1)$$

因为 n_i 和 n_{i+1} 在最佳路径上, 所以有

$$g^*(n_{i+1}) = g^*(n_i) + C(n_i, n_{i+1})$$

代入 (1) 式后有 $g^*(n_i) + h(n_i) \leq g^*(n_{i+1}) + h(n_{i+1})$

这个不等式对 P 上所有相邻的节点都合适, 若从 $i=1$ 到 $i=k-1$ 应用该不等式, 并利用传递性有

$$g^*(n_{i+1}) + h(n_{i+1}) \leq g^*(n_k) + h(n_k)$$

$$\text{即} \quad f(n_{i+1}) \leq g^*(n) + h(n) \quad (2)$$

另一方面, A^* 选 n 来扩展, 必有

$$f(n) = g(n) + h(n) \leq f(n_{i+1}) \quad (3)$$

比较 (2)、(3) 得 $g(n) \leq g^*(n)$, 但已知 $g(m) \geq g^*(m)$, 因此选 n 扩展时必有 $g(n) = g^*(n)$, 即找到了到达 n 的最佳路径。

〔证毕〕

定理 6: 若 $h(n)$ 满足单调限制, 则由 A^* 所扩展的节点序列, 其 f 值是非递减的, 即 $f(n_i) \leq f(n_j)$

证明: 单调限制条件有

$$h(n_i) - h(n_j) \leq C(n_i, n_j)$$

即 $f(n_i) - g(n_i) - f(n_j) + g(n_j) \leq C(n_i, n_j)$

$$f(n_i) - g(n_i) - f(n_j) + g(n_j) + C(n_i, n_j) \leq C(n_i, n_j)$$

$$f(n_i) - f(n_j) \leq 0. \quad \text{〔证毕〕}$$

根据定理 5, 在应用 A^* 算法时, 在第 6 步可不必进行节点的指针修正操作, 因而改善了 A^* 的效率。

另一方面我们从图 2.12 的例子看出, 因 h 不满足单调限制条件, 在扩展节点 n 时, 有可能还没有找到到达 n 的最佳路径, 因此该节点还会再次被放入 OPEN 中。为了使 A^* 算法少进行重复扩展, 可作如下改进。

由引理 2.2 知在搜索结束前, OPEN 表 内存 在 满足 $f(n) \leq f^*(s)$ 的节点。设 A^* 扩展了节点 n' , 则 $f(n') \leq f(n) \leq f^*(s)$, 再设搜索进程的某一时刻扩展的节点中, 扩展前 f 值最大的为节点 m , $f(m) \leq f^*(s)$ 。这时对在 OPEN 中的节点, 若 $f(n_i) < f(m)$, 则 n_i 必定被扩展几次, 因为 $f(n_i) < f^*(s)$ 必成立。因此 $n_i (f(n_i))$ 留在 OPEN 中, 目标节点 $t (f(t) = f^*(s))$ 就不能从 OPEN 中被取出来, 即使在扩展 m 之后, 扩展全部满足 $f(n_i) < f(m)$ 的节点也有作用。如果这样的节点有多个, 扩展的顺序不以 f 排列, 而以 g 的大小排列, 则在扩展期间, 同一个节点就 不会被扩展多次了。这样的搜索算法称为修正的 A^* 算法。

修正过程A

- ① OPEN:=(s), $f(s) = g(s) + h(s) = h(s)$, $f_m := 0$;
- ② LOOP: IF OPEN=() THEN EXIT(FAIL);
- ③ NEST:={ $n_i | f(n_i) < f_m$ }, NEST 给出 OPEN 中满足 $f < f_m$ 的节点集合。

```
IF NEST ≠ ( ) THEN n := n(min g_i)
ELSE n := FIRST(OPEN),  $f_m := f(n)$ ;
```

NEST 不空时, 取其中 g 最小者作为当前节点, 否则取 OPEN 的第一个为当前节点。

- ④
- ⋮
- ⑧ } 同过程 A

现在看一下用修正 A* 搜索图 2.12 的情况:

OPEN	f_m	CLOSED
初始化(s(0+20))	0	
1 (A(11+1) B(9+4) C(6+8) D(1+14))	20	(s(0+20))
2 (A(7+1) B(5+4) C(2+8))	20	(s(0+20) D(1+14))
3 (A(5+1) B(3+4))	20	(s(0+20) C(2+8) D(1+14))
4 (A(4+1))	20	(s(0+20) B(3+4) C(2+8))
5 (t(22+0))	20	D(1+14))
成功结束		

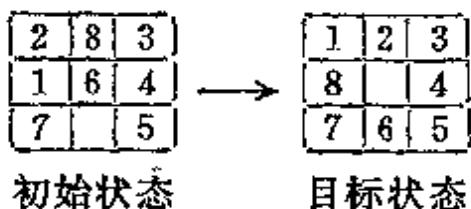
由 OPEN 和 CLOSED 中的状态看出, 修正后的算法减少了重复扩展的次数。一般情况下, 它比 A* 算法扩展节点的次数要少或相等。

8. A* 算法应用举例

A* 算法的理论意义在于给出了求解最佳解的条件 $h(n) \leq h^*(n)$ 。对给定的问题, 函数 $h^*(n)$ (n 是变量) 在问题有解的条件

下客观上是存在的，但在问题求解过程中不可能明确知道，因此对实际问题，能不能使所定义的启发函数满足下界范围条件？如果困难很大，那么 A* 算法的实际应用就会受到限制。下面将通过几个应用实例来说明这个问题。

(1) 八数码问题



对八数码问题，如果启发函数根据任意节点与目标之间的差异来定义，例如取 $h(n) = W(n)$ ，那么很容易看出，尽管我们对具体的 $h^*(n)$ 是多少很难确切知道，但根据“不在位”将牌个数这个估计，就能得出至少要移动 $W(n)$ 步才能达到目标，显然有 $W(n) \leq h^*(n)$ （假定为单位耗散的情况）。如果启发函数进一步考虑任意节点与目标之间距离的信息，例如取 $h(n) = P(n)$ ， $P(n)$ 定义为每一个将牌与其目标位置之间距离（不考虑夹在其间的将牌）的总和，那么同样能断定至少也要移动 $P(n)$ 步才能达到目标，因此有 $P(n) \leq h^*(n)$ 。图 2.13 给出 $h(n) = P(n)$ 时的搜索图，节点旁边还标出 $W(n)$ 和 $P(n)$ 启发函数值。由解路可给出 $g^*(n)$ 和 $h^*(n)$ 的值，由此可得最佳路径上的节点有 $f^* = g^* + h^* = 5$ 。

下面给出该八数码问题取不同启发函数，应用 A* 算法求得最佳解时所扩展和生成的节点数。

启发函数	$h(n) = 0$	$h(n) = W(n)$	$h(n) = P(n)$
扩展节点数	26	6	5
生成节点数	46	13	11

(2) M-C 问题

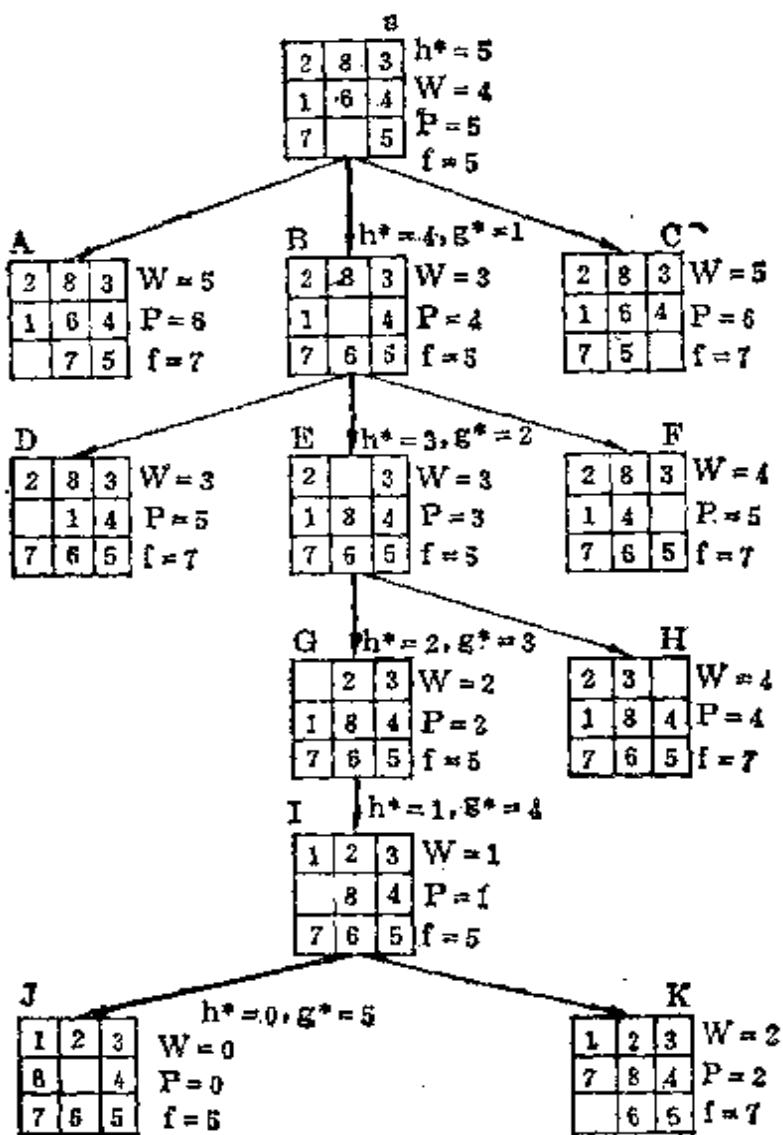


图 2.13 $h(n)=P(n)$ 的搜索树

设 $M=C=5$ ， $k \leq 3$ ，由分析可知至少要往返 10 次才可能把全部 M 和 C 摆渡到右岸，因此 M 和 C 的线性组合可作为启发函数的基本分量，此外还可以考虑有船与否对摆渡的影响。图 2.14 给出 $h(n)=0$ 、 $h(n)=M+C$ 、 $h(n)=M+C-2B$ 三种启发函数的搜索图（假定为单位耗散）。可以看出 $h(n)=0$ 的搜索图

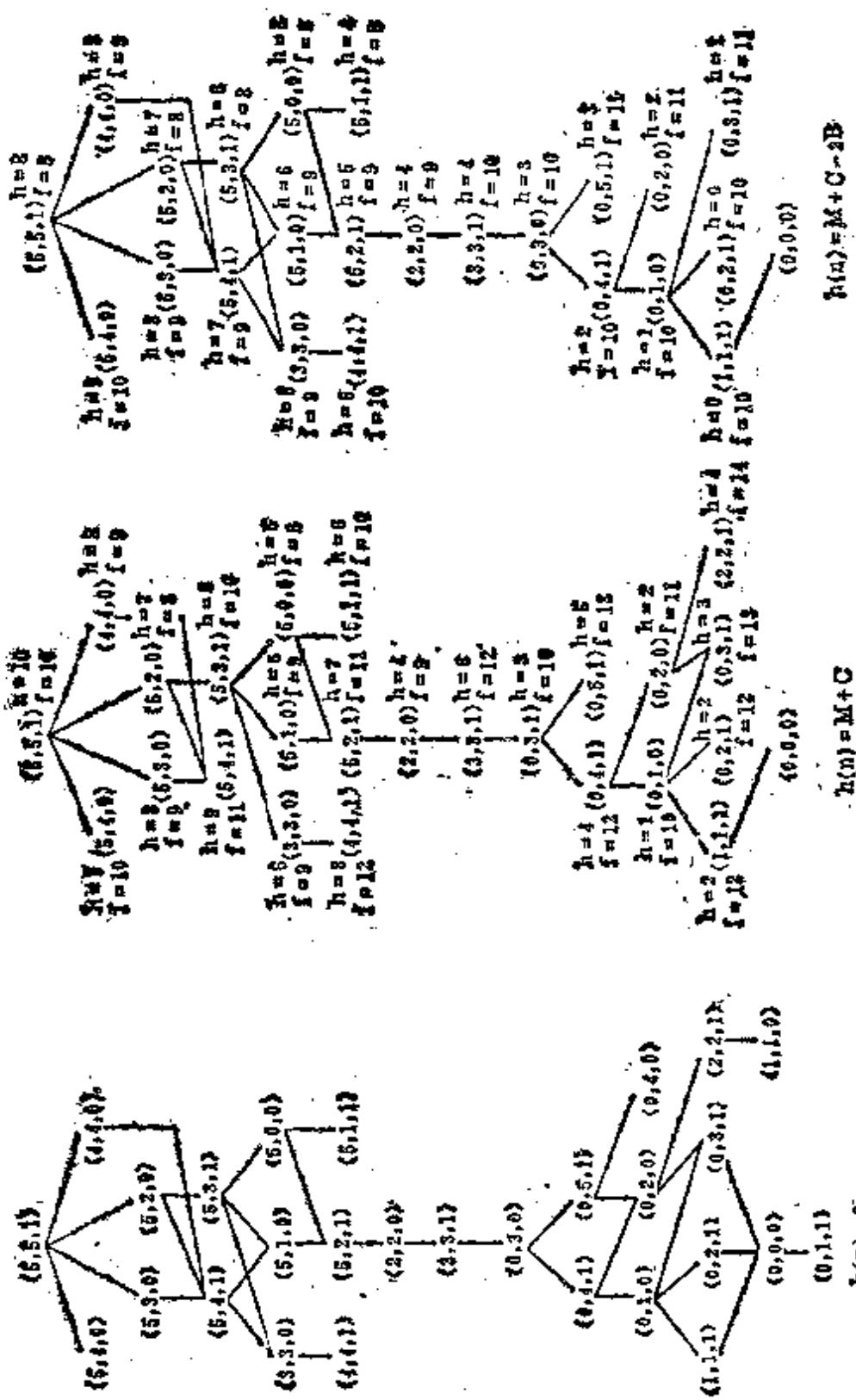


图 2.14 M-C 问题搜索图

就是该问题的状态空间图;取 $h(n) = M + C$ 不满足 $h(n) \leq h^*(n)$ 条件; 只有 $h(n) = M + C - 2B$ 可满足 $h(n) \leq h^*(n)$ 。

$M = C = 5$, $k \leq 3$ 的 $M - C$ 问题, 三种启发函数求解结果比较如下:

	$h(n) = 0$	$h(n) = M + C$	$h(n) = M + C - 2B$
生成节点数	23	23	23
扩展节点数	23	20	17
解路径耗散值	11	11	11

(3) 迷宫问题

迷宫图入口到出口有若干条通路, 求从入口至出口处最短路径的走法。

图 2.15 为一简单迷宫示意图及其平面坐标表示。以平面坐

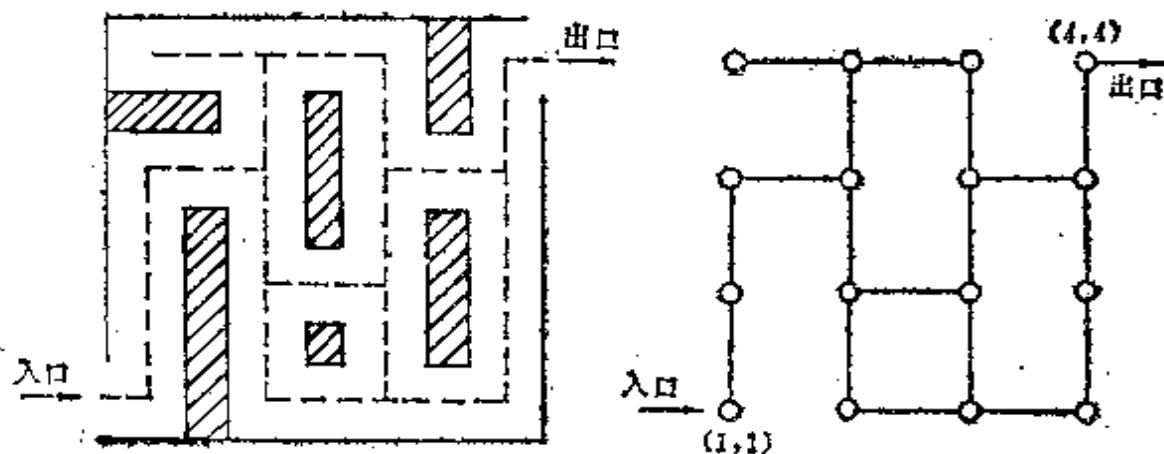


图 2.15 迷宫问题及其表示

用图来表示迷宫的通路时，问题的状态以所处的坐标位置来表示，即综合数据库定义为 (x, y) ， $1 \leq x, y \leq N$ ，则迷宫问题归结为求 $(1, 1) \rightarrow (4, 4)$ 的最短路径问题。

迷宫走法规定为向东、南、西、北前进一步，由此可得规则集简化形式如下：

$R_1: \text{if } (x, y) \text{ then } (x + 1, y)$

$R_2: \text{if } (x, y) \text{ then } (x, y - 1)$

$R_3: \text{if } (x, y) \text{ then } (x - 1, y)$

$R_4: \text{if } (x, y) \text{ then } (x, y + 1)$

对于这个简单例子，可给出状态空间如图 2.16 所示。

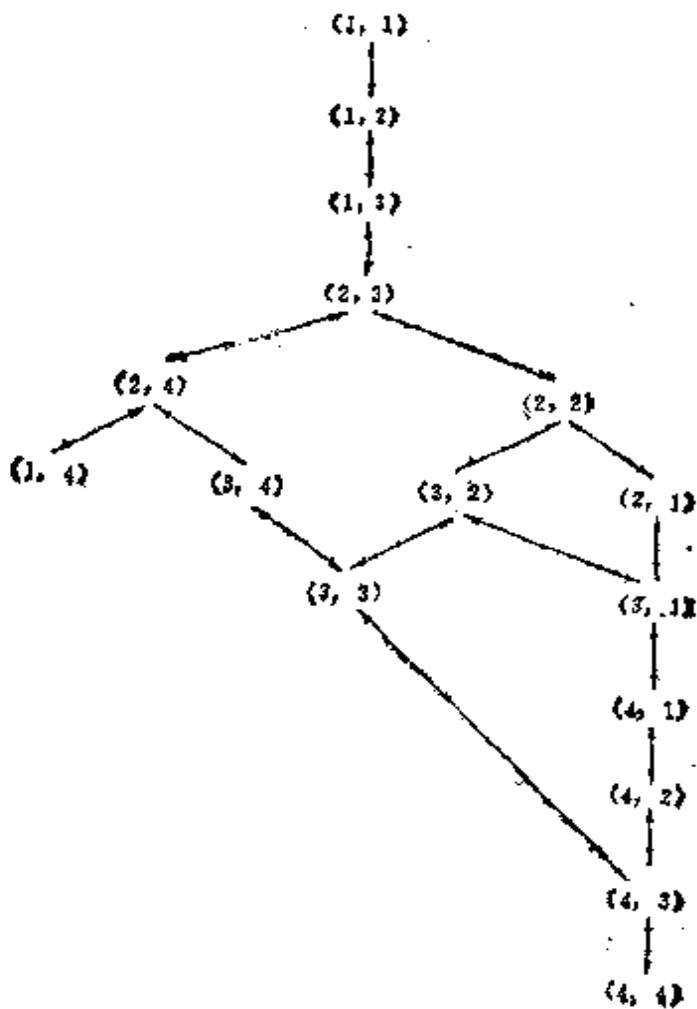


图 2.16 状态空间图

为求得最佳路径，可使用 A* 算法。假定搜索一步取单位耗散，则可定义

$$h(n) = |X_G - x_n| + |Y_G - y_n|$$

其中 (X_G, Y_G) 为目标点坐标， (x_n, y_n) 为节点 n 的坐标，显然有 $h(n) \leq h^*(n)$ 。取 $g(n) = d(n)$ ，有 $f(n) = d(n) + h(n)$ 。再设搜索过程在 OPEN 表中 $f(n)$ 相等时，以深度优先排序，则搜索图如图 2.17 所示。最短路径为 $((1, 1), (2, 3), (2, 4), (3, 4), (3, 3), (4, 3), (4, 4))$ 。

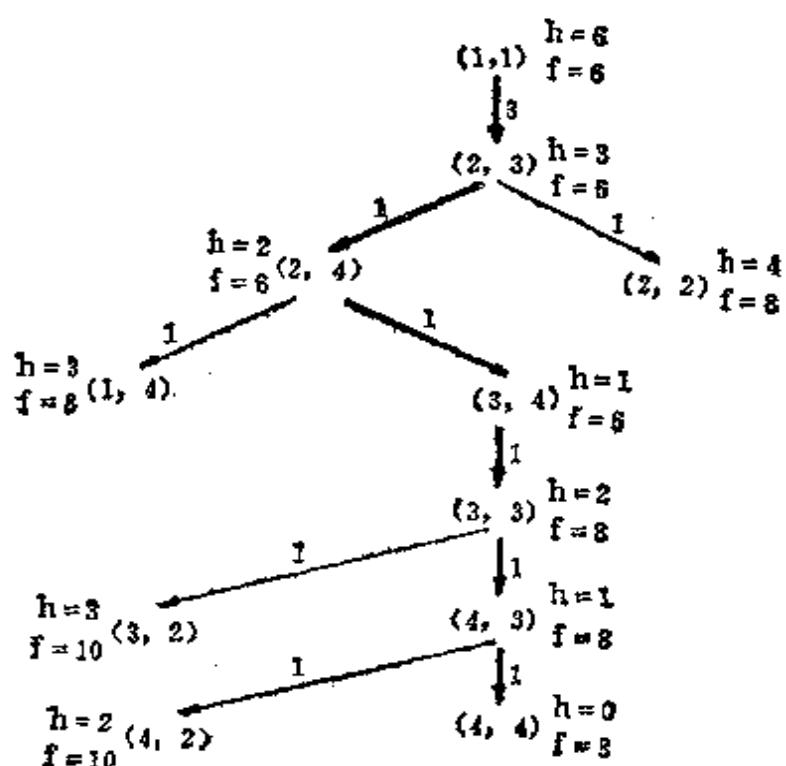


图 2.17 迷宫问题启发式搜索图

9. 评价函数的启发能力

首先我们通过八数码问题

2	1	6
4		8
7	5	3

 \longrightarrow

1	2	3
8		4
7	6	5

为例说明 A 算法的启发能力与选择启发函数 $h(n)$ 的关系。一般来说启发能力强，则搜索效率较高。有时选用不是 $h^*(n)$ 下界范围的 $h(n)$ 时，虽然会牺牲找到最佳解的性能，但可使启发能力得到改善，从而有利于求解一些较难的问题。

求解这个八数码问题，使用启发函数 $h(n) = P(n)$ 时，仍不能估计出交换相邻两个将牌位置难易程度的影响，为此可再引入 $S(n)$ 分量。 $S(n)$ 是对节点 n 中将牌排列顺序的计分值，规定对非中心位置的将牌，顺某一方向检查，若某一将牌后面跟的后继者和目标状态相应将牌的顺序相比不一致时，则该将牌估分取 2，一致时则估分取 0；对中心位置有将牌时估分取 1，无将牌时估分取 0；所有非中心位置每个将牌估分总和加上中心位置的估分值定义为 $S(n)$ 。依据这些启发信息，取 $h(n) = P(n) + 3S(n)$ 时，就是用 $f(n) = g(n) + P(n) + 3S(n)$ 来估计最佳路径的耗散值。 $f(n)$ 值小的节点，确能反映该节点愈有希望处于到达目标节点的最佳路径上。图 2.18 给出了该问题的搜索树，图中圆圈中的数字是 f 函数值，不带圈的数字表示扩展顺序。由图看出 $h(n)$ 函数不满足下界范围，但找到的是最小长度（18步）的解路径。

还有一个决定搜索算法启发能力的因素是涉及到计算启发函数的工作量，从被扩展的节点数最少的角度看， $h = h^*$ 最优，但这可能导致繁重的计算工作量。有时候一个不是 h^* 下界范围的 h 函数可能比起下界范围的 h 函数更容易计算，而且被扩展

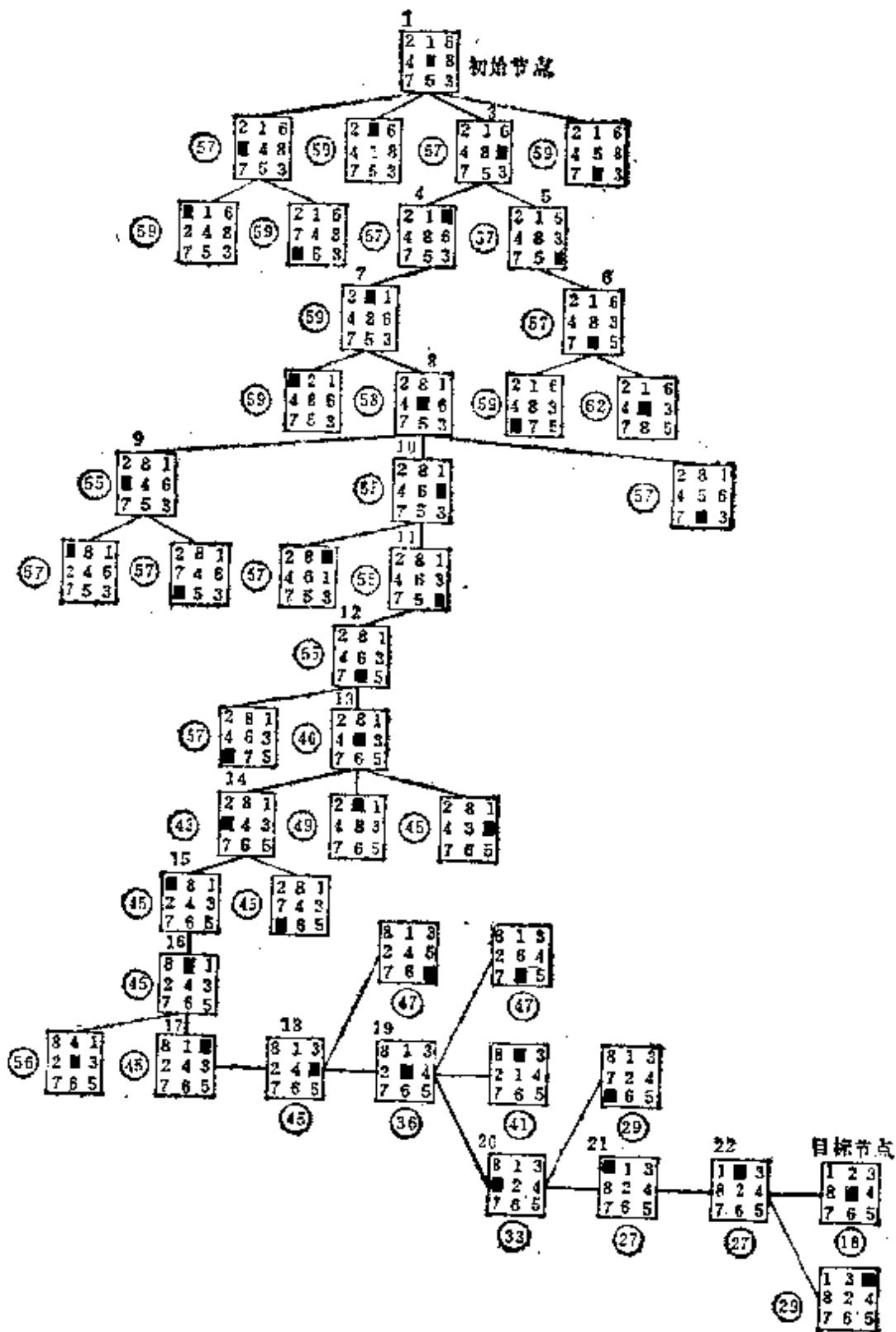


图 2.18 $b(n) = p(n) + 3s(n)$ 的搜索树

节点的总数可以减少，使启发能力加倍得到改善，虽然牺牲了可采纳性，但从启发能力的角度看仍是可取的。

在某些情况下，要改变启发能力，可以通过对 h 函数乘以加权系数的简单方法实现。当加权系数很大时， g 分量的作用相对减弱，因而可略去不计，这相当于在搜索期间任何阶段上，我们不在乎到目前为止所得到的路径耗散值，而只关心找到目标节点所需的剩余工作量，即可以使用 $f=h$ 作为评价函数来对 OPEN 表上的节点排序。但是另一方面为了保证最终能找到通向目标节点的路径，即便找到一条最小耗散的路径也不是根本上要求的，还是应当考虑 g 的作用。特别是当 h 不是一个理想的估计时，在 f 中把 g 考虑进去就是在搜索中添加一个宽度优先分量，从而保证了隐含图中不会有某些部分不被搜索到。而只扩展 h 值最小的节点，则会引起搜索过程扩展了一些靠不住的节点。

评价函数中， g 和 h 相对比例可通过选择 $f=g+wh$ 中 w 的大小加以控制。 w 是一个正数，很大的 w 值则会过份强调启发分量，而过小的 w 值则突出宽度优先的特征。经验证明使 w 值随搜索树中节点深度成反比变化，可提高搜索效率。即在深度浅的地方，搜索主要依赖于启发分量；而较深的地方，搜索逐渐变成宽度优先，以保证到达目标的某一条路径最终被找到。

总结以上讨论，得出影响算法 A 启发能力的三个重要因素是：

- (1) 路径的耗散值；
- (2) 求解路径时所扩展的节点数；
- (3) 计算 h 所需的工作量。

因此选择 h 函数时，应综合考虑这些因素以便使启发能力最大。

2.5 搜索算法讨论

1. 弱方法

人工智能范畴的一些问题都比较复杂，一般无法用直接求解的方法找到解答，因此通常都要藉助于搜索技术。前面几节讨论的搜索方法，其描述均与问题领域无关，如果把这些方法应用于特定问题的领域时，其效率依赖于该领域知识应用的情况，从我们举过的几个例子就可说明这个问题。但由于这些方法难于克服搜索过程的组合爆炸问题，因此在人工智能领域中，把这些方法统称为“弱方法”。这些搜索方法可用来求解不存在确定求解算法的某一类问题，或者虽然有某种求解算法，但复杂性很高，有不少均属 NP- 完全类的问题。为避免求解过程的组合爆炸，在搜索算法中引入启发性信息，多数情况能以较少的代价找到解，但并不能保证任何情况下都能获得解，这就是所谓“弱方法”的含义。当然如果引入强有力的启发信息，则求解过程就能显示出“强”的作用。下面我们来讨论用优选法求解极值这类问题时搜索过程的特点。

设状态是实数域 $[a, b]$ 上实值连续函数 f ，求该目标函数在何处取得极值及其大小。

在几何学中黄金分割法的思想是在区间 $[0, 1]$ 间取 0.618 和 0.382 两个特殊点来考虑问题：若 $f(0.382)$ 较优，则剪去 $[0.618, 1]$ 区段；若 $f(0.618)$ 较优，则剪去 $[0, 0.382]$ 区段；然后在新区间依此规则继续下去，直至函数 f 在某一点取得极值，这就是优选法的要点。显然目标函数 f 中包含了启发信息，下面给出该算法（应用该算法时先把 $[a, b]$ 通过变换转换成 $[0, 1]$ ）：

黄金分割法

① $x_1 := 0, x_2 := 1, x_3 := 0.382, x_4 := 0.618;$

赋变量初值

② LOOP1: IF $x_3 = x_4$ THEN EXIT (SUCCESS) ;

IF $f(x_3) > f(x_4)$ THEN GO LOOP2;

IF $f(x_3) < f(x_4)$ THEN GO LOOP3;

IF $f(x_3) = f(x_4)$ THEN GO LOOP2 \vee LOOP3;

可根据某种原则决定

③ LOOP2: $x_2 := x_4, x_3 := x_3, x_4 := x_1 + x_2 - x_3;$

④ GO LOOP1;

⑤ LOOP3: $x_1 := x_3, x_3 := x_4, x_4 := x_1 + x_2 - x_3;$

⑥ GO LOOP1;

由于计算中引入了极强的启发信息，因而获得最佳的搜索效果，可以证明 f 在 $[0, 1]$ 间具有单极值时， $f(x_3)$ 或 $f(x_4)$ 即为求得的极值点，而且求解过程搜索的点数是最少的。

前面我们讨论的几个搜索算法都属弱方法，实际上人工智能研究中提出属于这类的算法很多（如约束满足法，手段目的分析法等等），都可以用来求解某一特定问题，至于具体选择哪一种策略，很大程度上取决于问题的特征和实际要求。另外这些方法只提供了一种框架，对复杂问题只要能较好地运用特定问题的启发信息，就可能获得较好的搜索效果。

2. 搜索算法分析

算法分析主要要回答这些方法执行的效果怎样，找到的解其优劣程度如何。在一般的计算机科学领域中，主要强调对算法进行数学分析，如严格数学分析遇到困难，则采取运行一组经过精心挑选的问题，再对其执行情况作统计分析。而人工智能领域研究这个问题的途径则采取将算法用某种语言具体实现，然后运行某个智能问题的典型实例，并观察其表现行为来进行分析比较。

这主要是由于人工智能问题比较复杂，通常不容易对一过程是否可行作出令人信服的分析证明。此外，有时甚至无法把问题的值域描述清楚，因而也难于对程序行为作统计分析。再一点就是人工智能是一门实验科学，实践是目前主要的研究途径。

搜索过程最基本的一个分析是对深度为 D 、分支因数为 B 的一棵完全树的节点数（为 B^D ）进行讨论。显然如果一过程执行的时间随问题的规模变大而成指数增长时，则该过程无实用意义。因此要研究改进穷举搜索的各种方法，并通过所得到的搜索时间上界，来和穷举法比较改善的程度。目前优于穷举法的若干方法有三种情况：

（1）能保证找到的解与穷举法所得结果一样好，但耗时较少。这类方法的问题是能否给出某一方法具体有多快。

（2）对问题的一些实例，耗费时间和穷举法一样，但对另一些实例则较穷举法好得多。这类方法的问题是运行一组一般问题，期望有多快。

（3）得到的解比穷举法结果较差。问题是要在给定时间内找到解，这个解与最佳解之间有多大差别。

此外对 NP- 完全类问题，已知若干非确定型（即每次可得到任意数目的路径数）多项式时间的算法，但所有已知的确定性算法都是指数型的。可以证明，NP- 完全类中的若干问题在下述意义上彼此等价：如果能找到其中一个问题的一种确定型多项式时间算法，则该算法可应用于所有的问题。

综上所述，求解人工智能问题的一个途径是企图以多项式时间求解 NP- 完全类问题。实现这一点可有两种选择：一是寻找平均角度看执行较快的一些算法，即使在最坏的情况下不慢也行；另一是寻找近似算法，使能在可接受的时间限度内获得满意的解答。

分支界限法实际上是第一种选择的实例，对这种策略有如下

评述：

(1) 各种选择按完美排序进行时，最好情况下其性能有多好。

(2) 各种选择按不良排序（从最坏到最好）进行时，最差情况下其性能有多好。结果显然，其搜索节点数同穷举法，此外还必须附加跟踪当前约束及进行无畏的比较工作量。

(3) 各种选择按某种随机过程的排序进行时，平均情况下其性能有多好。

(4) 各种选择按应用于一组特定问题的启发函数排序进行时，在实际世界中，平均角度看其性能有多好。通常这个结果优于真正随机世界的平均情况。

总的来说，人们愿意接受良好的平均时间性能，但即使如此也未必能做得满意，就连分支界限法的平均时间也是指数型的($\sim 1.2e^N$, N 为问题的规模)。因此有时也得牺牲完美解的要求并接受近似的解法。一种进一步改进的分支界限法，用于求解旅行商问题称为最近邻居算法就是求满意解的一个实例，通过分析可以得到所需的时间与 N^2 成比例。

至于 A 算法既是“寻找平均较快”策略的例子，又是求满意解策略的例子，关键是启发函数 h 的选取问题。至于更深入的讨论可参阅有关文献。

3. 数字魔方问题求解的搜索策略

数字魔方游戏已有悠久的历史，是古代数学家、哲学家、神学家、占星家等探索的问题。现在来看一个 1750 年 Benjamin Franklin 编造的一个数字魔方。在 8×8 的棋盘方格上，填入 1—64 的数字，其结果如图 2.19 所示。其主要特性是每行或每列 8 个数字的总和为 260。再进一步分析发现任一平行或半列 4 个数字的总和为 130；任一拐弯对角线 8 个数字的总

52	61	4	13	20	29	36	45
14	3	62	51	46	35	30	19
53	60	5	12	21	28	37	44
11	6	59	54	43	38	27	22
58	58	7	10	23	26	39	42
9	8	57	56	41	40	25	24
50	63	2	15	18	31	34	47
16	1	64	49	48	33	32	17

图 2.19 Benjamin Franklin 数字魔方

和也是 260 (如 16, 63, 57, 10, 23, 40, 34, 17 或 50, 8, 7, 54, 43, 26, 25, 47 两条拐弯对角线)；4 角的 4 个数字 (52, 45, 16, 17) 与中心的 4 个数字 (54, 43, 10, 23) 也是 260；任意一个 2×2 的子魔方 4 个数字之和是 130 (如 52, 61, 14, 3 或 23, 26, 41, 40)，还可给出几个类似的性质。Benjamin, Franklin 还研究了魔圆的构造问题，9 个同心圆等分为 8 个扇区，将 12—75 共 64 个数字填入空格中，如图 2.20 所示。其性质是：任一同心圆周格子上的 8 个数字之和加上中心圆的数字 (为起始数 12) 总和为 360；任一径向 8 个数字之和加上中心圆的数字总和亦为 360 (圆周的度数)。更为惊讶的是任一径向半列的 4 个相邻格子数字和加上 6，其总和为 180；由水平直径分割后任一半圆数字和加上 6，其总和仍为 180。由此看出构造这种数字魔方或魔圆并满足若干性质是一个极为复杂的搜索问题。

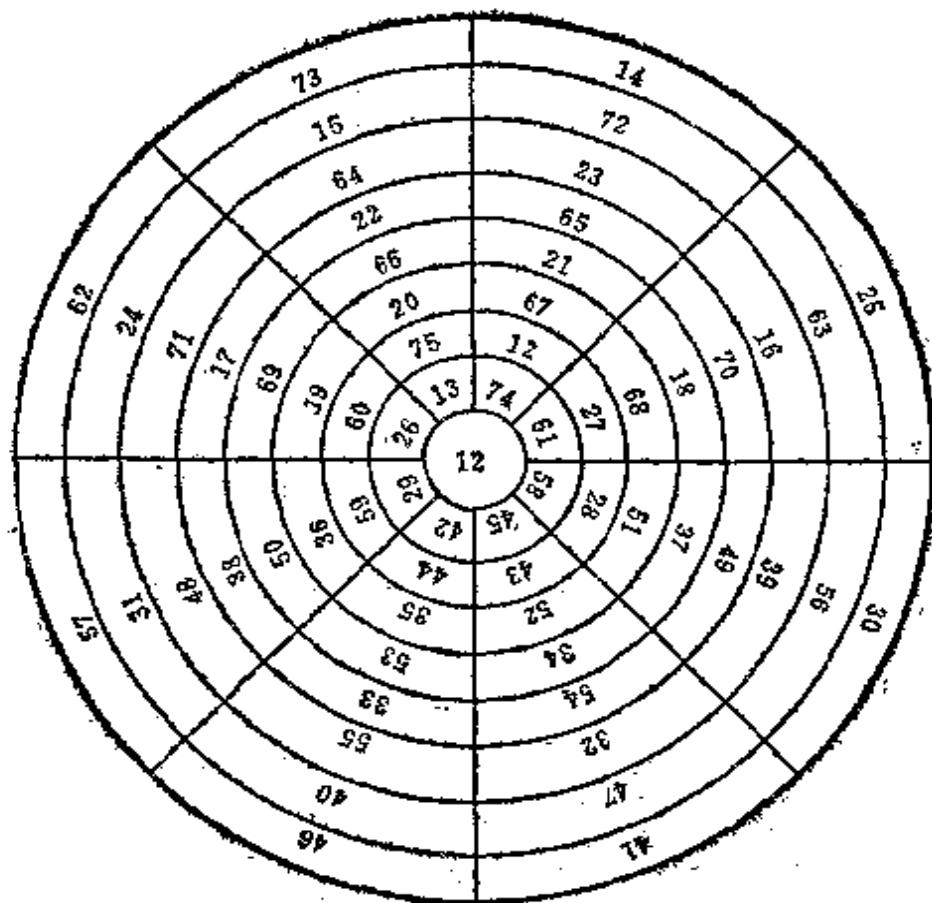


图 2.20 Benjamin Franklin 数字魔圆

下面来讨论一个最简单的 3×3 数字魔方问题。将 1—9 共 9 个数码填入魔方格使行、列和对角线数码总和相等。对于奇数阶魔方问题，数学家们已构造出一个极为巧妙的算法，不花费任何多余的搜索就找到最佳解路。这个算法的要点如下：

N 阶（奇次）魔方算法：

① $D := 1, P(D, (x = \frac{[N]}{2}, y = N))$ ；函数 P 把最小数字置于

顶行中心处，每一方格用坐标标记，如图 2.21。

② LOOP: $D := D + 1, x := x + 1, y := y + 1$ ；

⑤ GO LOOP,

该算法应用于 3×3 数字魔方问题,其搜索图如图2.22所示,从搜索过程看出,构造算法的基本启发信息是把数码等分成N组,每一组N个数码放置原则是每一个数码必须处在不同行不同列的方格上,这样搭配就可能使行、列和对角线的数字和相接近乃至完全相等。这是最有希望获得目标要求的搜索方向,这样就把许多没有希望的路径抛弃,从而大大提高了搜索效果。

进一步分析数字魔方的要求,还可以给出若干有用的启发信息,例如N(奇次)阶魔方,行、列或对角的总和值 $S=N\left(\frac{1+N^2}{2}\right)$,在 $C_N^{N^2}$ 种的数字组合中,只有满足数字总和为S的 $2N+2$ 组排法才可能构成解。其中具有公共元的4组数字应排列在中心行、中心列

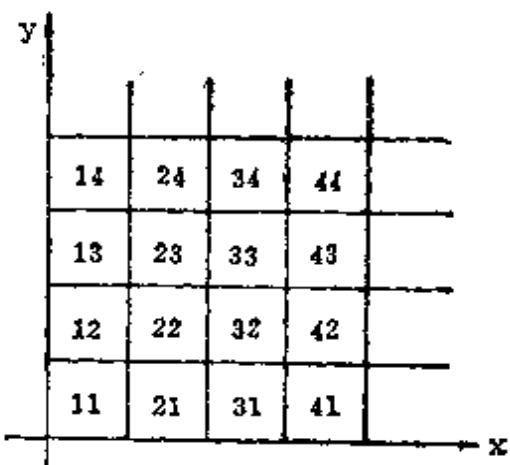


图 2.21 魔方坐标图

和对角上，且公共元的那个数码必定处在中心格位置上，还有3

组共有的那些数码必定处在四角位置上等等。利用这些信息引导搜索，可求解任意阶数字魔方问题。

对于任意偶次阶的数字魔方问题，也有简单的求解算法，读者可从程序设计的参考书找到，这里不再介绍。

4. 搜索算法的研究工作

A*算法是N.J.Nilsson70年代初的研究成果，是从函数的观点来讨论搜索问题，并在理论上取得若干结果。但A*算法不能完全克服“指数爆炸”的困难。

70年代末J.Pearl从概率观点研究了启发式估计的精度同A*算法平均复杂性的关系。Pearl假设如下的概率搜索空间：一个一致的 m^n 枝树G，

在深度d处有一个唯一的目标节点 G_d ，其位置事先并不知道。

设估计量 $h^*(n)$ 是在 $[0, h^*(n)]$ 区间中的随机变量，由分布函数 $F_{h(n)}(x) = P[h(n) \leq x]$ 来描述； $E(Z)$ 表示用A*算法求到目标 G_d 时所展开的节点平均个数，并称之为A*的平均复杂性。若 $h(n)$ 满足

$$P\left[\frac{h^*(n) - h(n)}{h^*(n)} > \epsilon\right] > \frac{1}{m}, \quad \epsilon > 0$$

则有 $E(Z) \sim O(e^{cd})$ ， $c > 0$

80年代初，张钹、张铃提出把启发式搜索看成某种随机取样的过程，从而将统计推断方法引入启发式搜索。把各种统计推断方法；如序贯概率比检验法（SPRT），均值固定宽度信度区间渐近序贯法（ASM）等，同启发式搜索算法相结合，得到一种称为SA（统计启发式）的搜索算法。该算法在一定假设条件下，能以概率为一找到目标，且其平均复杂性为 $O(d\ln d)$ 或 $O(d\ln^2 d)$ 。而且证明其所需的条件比使A*搜索为多项式复杂性的条件为弱。

总之搜索策略是人工智能研究的核心问题之一，已有许多成熟的结果，并在解决人工智能的有关问题中得到广泛应用。但目前仍有若干深入的问题有待发展，特别是结合实际问题，探索有效实用的策略仍是一个研究和开发的工作，还应当给予足够的重视。

2.6 小 结

1. 搜索策略是产生式系统最主要的组成部分，其主要任务是决定如何选取产生式规则，判定是否满足目标条件，以及记录解序列。本章较详细讨论了回溯策略算法，图搜索算法，最佳搜索算法A*及其理论结果。还通过若干实际例题，深入讨论了搜索算法的应用及研究工作中所提出的一些问题。

2. 利用递归过程描述回溯控制机制简单有效。具体算法可根据问题的性质，建立必要的回溯点来实现回溯的控制。当给定问题有解时，回溯法一定能求得解，但不能保证获得最佳解。如果利用问题有关知识来引导搜索过程，则可减少回溯次数，提高搜索效果。

3. 启发式图搜索策略是人工智能系统中最常用的控制策略，它是利用问题领域拥有的启发信息来引导搜索过程，达到减少搜

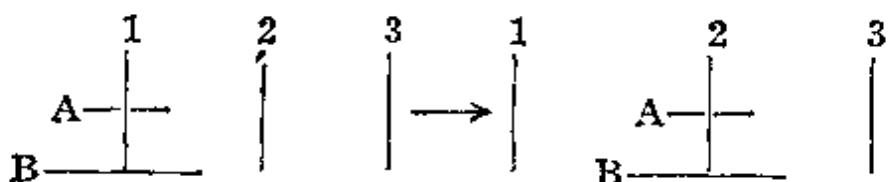
索范围，降低问题复杂度的目的。根据图搜索算法理论建立的最佳搜索算法 A^* ，只要使启发函数 $h(n) \leq h^*(n)$ ，就能保证找到最佳解。对一些比较简单实际问题，可以建立一个数值函数 $h(n) \leq h^*(n)$ ，但对复杂实际人工智能问题较难做到这一点。此外 A^* 算法的问题复杂性与 $h(n)$ 的选取有关，一般情况下， A^* 算法没有解决指数爆炸的问题。

4. 根据问题的知识建立启发函数 $h(n)$ 是解决实际问题要碰到的重要问题，通常可以根据一个节点处在最佳路径上的概率，求出任意一个节点与目标节点集之间的距离度量或差异度量或根据格局（或状态）特征的记分法，来建立实用的 $h(n)$ 函数。

5. 搜索策略是人工智能研究的核心课题之一，已有许多成熟成果并在实践中得到广泛应用。实际应用中的研究工作，主要是解决算法复杂性的问题，探索有效和实用的搜索策略仍然是很有实际意义的工作。

习 题

2.1 用回溯策略求解如下所示二阶梵塔问题，画出搜索过程的状态变化示意图。



对每个状态规定的操作顺序为：先搬 1 柱的盘，放的顺序是先 2 柱后 3 柱；再搬 2 柱的盘，放的顺序是先 3 柱后 1 柱；最后搬 3 柱的盘，放的顺序是先 1 柱后 2 柱。

2.2 滑动积木块游戏的棋盘结构及某一种将牌的初始排列结构如下：

B	B	B	W	W	W	E
---	---	---	---	---	---	---

其中 B 表示黑色将牌， W 表示白色将牌， E 表示空格。游戏的规定走法是：

(1) 任意一个将牌可以移入相邻的空格，规定其耗散值为 1；

(2) 任意一个将牌可相隔 1 个或 2 个其他的将牌跳入空格，规定其耗值等于跳过将牌的数目；

游戏要达到的目标是使所有白将牌都处在黑将牌的左边（左边有无空格均可）。对这个问题，定义一个启发函数 $h(n)$ ，并给出利用这个启发函数用算法 A 求解时所产生的搜索树。你能否辨别这个 $h(n)$ 是否满足下界范围？在你的搜索树中，对所有的节点满足不满足单调限制？

2.3 对 1.4 节中的旅行商问题，定义两个 h 函数（非零），并给出利用这两个启发函数用算法 A 求解 1.4 节中的五城市问题。讨论这两个函数是否都在 h^* 的下界范围及求解结果。

2.4 2.1 节四皇后问题表述中，设应用每一条规则的耗散值均为 1，试描述这个问题 h^* 函数的一般特征。你是否认为任何 h 函数对引导搜索都是有用的？

2.5 对 $N=5$, $k \leq 3$ 的 M-C 问题，定义两个 h 函数（非零），并给出用这两个启发函数的 A 算法搜索图。讨论用这两个启发函数求解该问题时是否得到最佳解。

2.6 证明 OPEN 表上具有 $f(n) < f^*(s)$ 的任何节点 n ，最终都将被 A^* 选择去扩展。

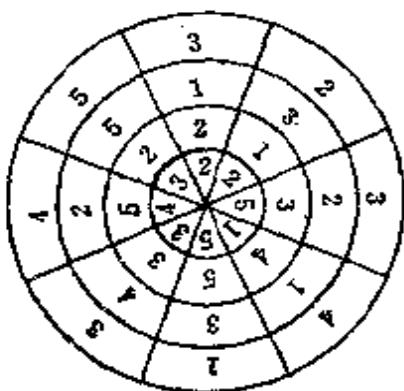
2.7 如果算法 A^* 从 OPEN 表中去掉任一节点 n ，对 n 有 $f(n) > F$ (F 处在 $f^*(s)$ 的上界范围)，试说明为什么算法 A^*

仍然是可采纳的。

2.8 用算法 A 逆向求解图2.7中的八数码问题，评价函数仍定义为 $f(n) = d(n) + w(n)$ 。逆向搜索在什么地方和正向搜索相会。

2.9 讨论一个 h 函数在搜索期间可以得到改善的几种方法。

2.10 四个同心圆盘的扇区数字如图所示，每个圆盘可单独转动。问如何转动圆盘使得八个径向的 4 个数字和均为 12。



2.11 在 3×3 的九宫格内，用 $1, 2, \dots, 9$ 的九个数字填入九宫内，使得每行数字组成的十进制数平方根为整数。

试用启发式搜索算法求解，分析问题空间的规模和有用的启发信息，

给出求解的搜索简图。

2.12 一个数码管由七段组成，用七段中某些段的亮与不亮可分别显示 0—9 这十个数字。问能否对这十个数字给出一种排列，使得每相邻两个数字之间的转换，只能是打开几个亮段或关闭几个亮段，而不能同时有打开的亮段，又有关闭的亮段。试用产生式系统求解该问题。

第三章 可分解产生式系 统的搜索策略

在第一章讨论可分解产生式系统时介绍了一种与或树的表示结构。本章将进一步讨论与或图（树）的搜索策略，还要讨论用于博弈系统的图搜索技术。

3.1 与或图的搜索

可分解产生式系统中提到的与或树表示，其中加到每一个节点上“与”或“或”的标记是取决于该节点对其父节点的关系。如复合数据库分解后拥有一组“与”关系的后继节点，而分量数据库经可应用规则作用后，生成一组“或”关系的后继节点。

通常我们要讨论与或图的一般情况（与或树是其特例），即应用不同的规则序列可能会生成出相同的数据库。例如某一个分量数据库标记的节点，可以是某一个已被分解的复合数据库的一个子节点，应称它为“与”节点，但这个分量数据库标记的节点也可以是另一个分量数据库使用某一规则之后得到的子节点，那么它又是“或”节点。为了避免混淆不清，通常不把与或图中节点叫做与节点或者或节点，而是引入一个适合于与或图的更一般的标记，而在称谓上沿用习惯，仍把这种结构称作与或图。当然在讨论与或树时仍继续用“与”、“或”节点的称呼。

图 3.1 给出一个简单的与或图例子，下面就来说明它的标记方法。

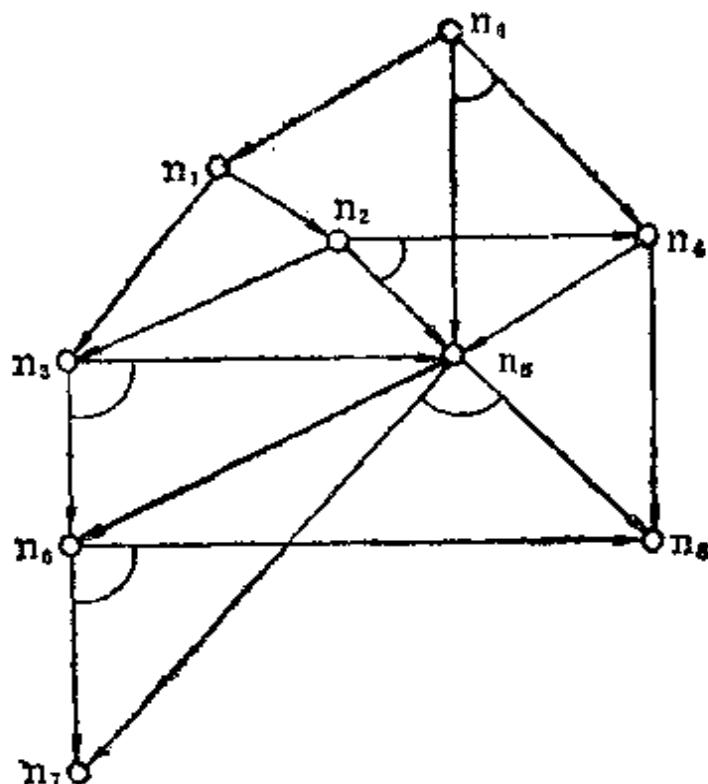


图 3.1 与或图

这个图也称做超图，节点间是用超弧线（或连接符）来连接，如一个父节点和一组后继节点的关系可用若干个连接符来标记，并规定 k -连接符是从一个父节点指向一组 k 个后继节点的节点集。这时若节点间全是 1 -连接符（相当于一条弧线）连接，显然这就是一般图的标记法，得到的就是与或图的特例——普通图。

从图 3.1 可以看出，节点 n_8 有两个连接符：1-连接符指向节点 n_1 ；2-连接符指向节点集 $\{n_4, n_5\}$ 。该 2-连接符还用一小段圆弧括起来（对 $k > 1$ 的 k -连接符都应有小圆弧标记），以表示子节点间与的关系。可以看出这种标记法在节点间具有明确的关系。显然若用原先的术语，则对父节点 n_8 来说， n_4, n_5 是两个与节点，而 n_1 可称为或节点；而 n_8 既是 n_5 的一个与节点，又是 n_4 的一个或节点，混淆难于避免。另外也把图中没有任何

父节点的节点叫根节点，没有任何后继节点的节点叫端节点或叶节点。

一个可分解产生式系统规定了一个隐含的与或图，初始数据库对应于图中的初始节点，它有一个外向的连接符连到它的一组后继节点，每个后继节点分别对应初始数据库中的一个分量；可应用于分量数据库的每条产生式规则，也对应于一个连接符，它指向的节点则相应于应用规则后生成的数据库，满足产生式系统结束条件的数据库是对应的一组终节点；产生式系统的任务是搜索从初始节点到一组终节点集 N 的一个解图。

下面就解图及其耗散值的概念和定义、能解不能解节点的定义作些说明。

与或图中某一个节点 n 到节点集 N 的一个解图类似于普通图中的一条解路径。解图的求法是：从节点 n 开始，正确选择一个外向连接符，再从该连接符所指的每一个后继节点出发，继续选一个外向连接符，如此进行下去直到由此产生的每一个后继节点成为集合 N 中的一个元素为止。图 3.2 给出 $n_0 \rightarrow \{n_1, n_2\}$ 的三个解图。

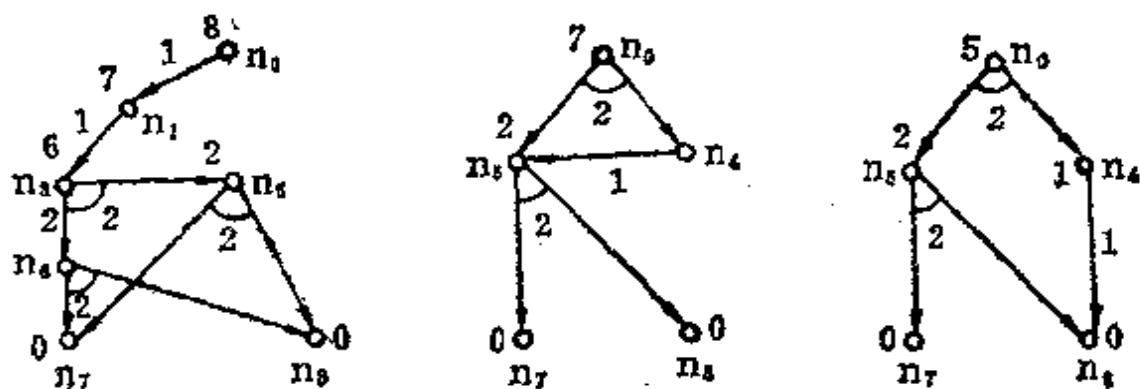


图 3.2 三个解图

在与或图是无环（即不存在这样的节点——其后继节点同时

又是它的祖先) 的假定下, 解图可递归定义如下:

定义: 一个与或图 G 中, 从节点 n 到节点集 N 的解图记为 G' , G' 是 G 的子图。

① 若 n 是 N 的一个元素, 则 G' 由单一节点组成;

② 若 n 有一个指向节点 $\{n_1, \dots, n_k\}$ 的外向连接符 K , 使得从每一个 n_i 到 N 有一个解图 ($i=1, \dots, k$), 则 G' 由节点 n , 连接符 K , 及 $\{n_1, \dots, n_k\}$ 中的每一个节点到 N 的解图所组成;

③ 否则 n 到 N 不存在解图。

在搜索解图的过程中, 还须要进行耗散值的计算。设连接符的耗散值规定为: k -连接符的耗散值 = k , 若解图的耗散值记为 $k(n, N)$, 则可递归计算如下:

① 若 n 是 N 的一个元素, 则 $k(n, N) = 0$;

② 若 n 有一个外向连接符指向后继节点 $\{n_1, \dots, n_i\}$, 并设该连接符的耗散值为 C_n , 则

$$k(n, N) = C_n + k(n_1, N) + \dots + k(n_i, N)$$

根据这个定义, 图 3.2 三个解图的耗散值计算结果分别为 8, 7 和 5。具有最小耗散值的解图称为最佳解图, 其值也用 $h^*(n)$ 标记, 上例中 $h^*(n_0) = 5$ 。

此外搜索过程还要标记能解节点 (SOLVED), 为此给出如下定义:

能解节点 (SOLVED):

① 终节点是能解节点;

② 若非终节点有“或”子节点时, 当且仅当其子节点至少有一能解, 该非终节点才能解;

③ 若非终节点有“与”子节点时, 当且仅当其子节点均能解, 该非终节点才能解。

不能解节点 (UNSOLVED):

① 没有后裔的非终节点是不能解节点;

- ② 若非终节点有“或”子节点时，当且仅当所有子节点均不能解时，该非终节点才不能解；
- ③ 若非终节点有“与”子节点时，当至少有一子节点不能解时，该非终节点才不能解。

3.2 与或图的启发式搜索算法AO*

启发式的与或图搜索过程和普通图类似，也是通过评价函数 $f(n)$ 来引导搜索过程。由于搜索的是一个解图，因而考察待扩展节点 n 的历史情况时，已不是从初始节点到 n 这条路径的 $g(n)$ 值就能说明问题，因此只考虑 $h(n)$ 这个分量，并企图通过 $h(n)$ 对 $h^*(n)$ 进行估计。下面首先讨论 AO* 算法本身，然后再通过示例说明搜索过程以及与 A* 算法的某些差别。

1. AO* 算法

过程 AO* :

- ① 建立一个搜索图 G , $G := s$, 计算 $q(s) = h(s)$, IF GOAL(s) THEN $M(s, SOLVED)$; 开始时图 G 只包括 s ，耗散值估计为 $h^*(s)$ ，若 s 是终节点，则标记上能解。
- ② Until s 已标记上 $SOLVED$, do:
- ③ begin
- ④ $G' := FIND(G)$; 根据连接符标记（指针）找出一个待扩展的局部解图 G' 。
- ⑤ $n = G'$ 中的任一非终节点；选一个非终节点作为当前节点。
- ⑥ EXPAND(n), 生成子节点集 $\{n_i\}$, $G := ADD(\{n_i\}, G)$, 计算 $q(n_i) = h(n_i)$, 其中 $n_i \notin G$,
IF GOAL(n_i) THEN $M(n_i, SOLVED)$; 把 n 的子节

点添加到 G 中，对 G 中未出现的子节点计算耗散值，若有终节点则加能解标记。

⑦ $S := \{n\}$ ，建立含 n 的单一节点集合 S 。

⑧ Until S 为空， do:

⑨ begin

⑩ REMOVE(m, S)， $m \notin \{S\}$ ，这个 m 的子节点 m 应不在 S 中。

⑪ •修改 m 的耗散值 $q(m)$ ，

对 m 指向节点集 $\{n_{1i}, n_{2i}, \dots, n_{ki}\}$ 的每一个连接符 i 分别计算 q_i ，

$$q_i(m) = C_i + q(n_{1i}) + \dots + q(n_{ki}),$$

$q(m) := \min q_i(m)$ ，对 m 的 i 个连接符，取计算结果最小的那个耗散值为 $q(m)$ 。

•加指针到 $\min q_i(m)$ 的连接符上，或把指针修改到 $\min q_i(m)$ 的连接符上，即原来指针与新确定的不一致时应删去。

•IF $M(n_{ij}, SOLVED) \rightarrow M(m, SOLVED)$ ，若该连接符的所有子节点都是能解的，则 m 也标上能解。

⑫ IF $M(m, SOLVED) \vee (q(m) \neq q_0(m))$

THEN ADD(m, S)， m 能解或修正的耗散值与原先估算 q_0 不同，则把 m 的所有先辈节点 m ，添加到 S 中。

⑬ end

⑭ end

这个算法可划分成两个操作阶段：第一阶段是4—6步，完成自顶向下的图生成操作，先通过有标记的连接符，找到目前为止最好的一个局部解图，然后对其中一个非终节点进行扩展，并对其后继节点赋估计耗散值和加能解标记。第二阶段是7—12步，完成自下向上的耗散值修正计算，连接符（即指针）的标记以及节点的能解标记。

耗散值的修正从刚被扩展的节点 n 开始, 其修正耗散值 $q(n)$ 取估计 $h^*(n)$ 的所有值中最小的一个, 然后根据耗散值递归计算公式逐级向上修正其先辈节点的耗散值, 只有下层节点耗散值修正后, 才可能影响上一层节点的耗散值, 因此必须自底向上一直修正到初始节点。这由算法中的内循环过程完成, 下面就用图 3.1 的例子来说明算法的应用过程。

2. 算法应用举例

设某个问题的状态空间如图 3.1 所示, 并定义了某个启发函数 $h(n)$, 我们来看一看解图的搜索过程。

为了使用方便, 将 $h(n)$ 函数对图 3.1 中各节点的假想估值先列写如下 (实际应用中是节点生成出来之后才根据 $h(n)$ 定义式计算) :

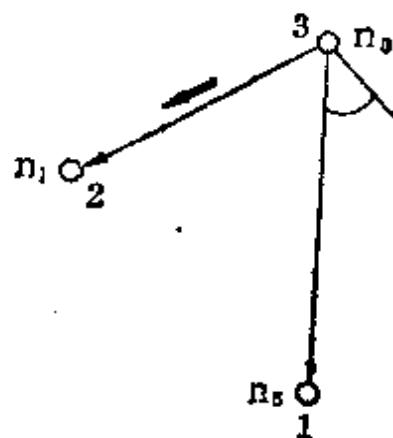
$h(n_0) = 3, h(n_1) = 2, h(n_2) = 4, h(n_3) = 4, h(n_4) = 1,$
 $h(n_5) = 1, h(n_6) = 2, h(n_7) = h(n_8) = 0$ (目标节点)。

此外仍假设 k -连接符的耗散值 = k 。

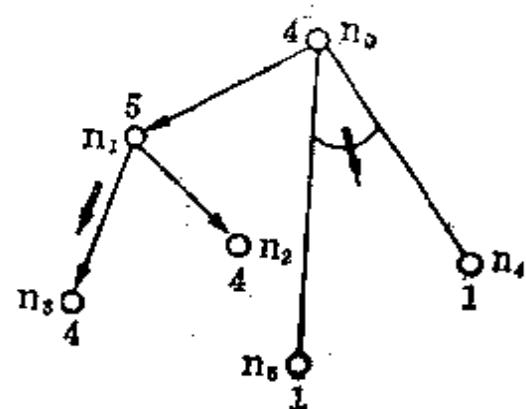
应用 AO* 算法求解, 经过 4 个大循环后就找到解图, 图 3.3 给出这 4 个循环之后的搜索结果。第四个循环后算法结束, 这时带指针的连接符就给出了所得到的解图, n_0 给出的修正耗散值 $q(n_0) = 5$ 就是解图的实际耗散值。该例中显然找到了具有最小耗散值的解图。

3. 算法应用的若干问题

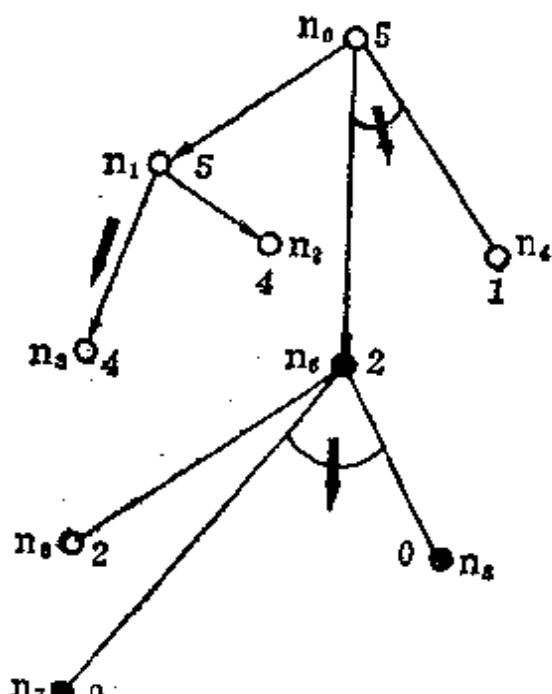
(1) 在第 6 步扩展节点 n 时, 若不存在后继节点 (即陷入死胡同), 则可在第 11 步中对 m (即 n) 赋一个高的 q 值, 这个高的 q 值会依次传递到 s , 使得含有节点 n 的子图具有高的 $q(s)$, 从而排除了被当作候选局部解图的可能性。



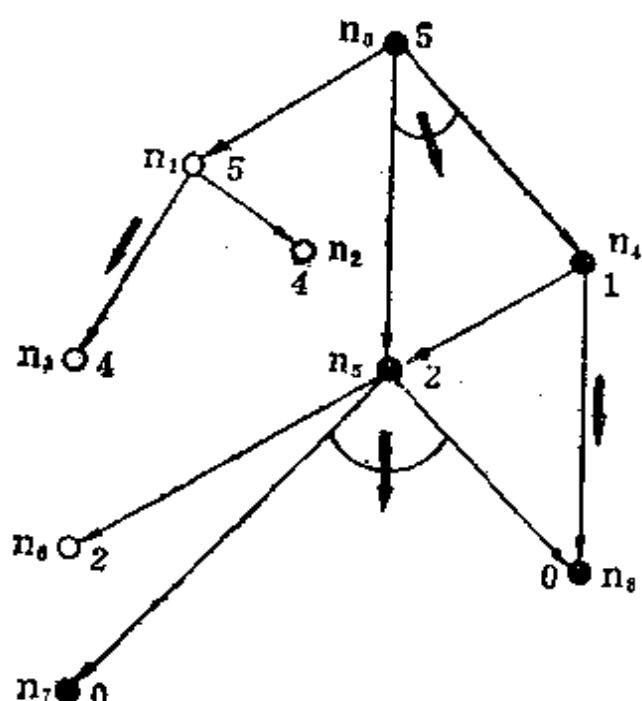
(a)



(b)



(c)



(d)

图 3.3 AO* 算法 4 个循环的搜索图

(a) 一次循环之后 (b) 两次循环之后
 (c) 三次循环之后 (d) 四次循环之后

(2) 第5步中怎样选出 G' 中的一个非终节点来扩展呢？一般可以选一个最可能导致该局部解图耗散值发生较大变化的那个节点先扩展，因为选这个节点先扩展，会促使及时修改局部解图的标记。

(3) 同 A 算法类似，若 $s \rightarrow N$ 集存在解图，当 $h(n) \leq h^*(n)$ 且 $h(n)$ 满足单调限制条件时，则 AO^* 一定能找到最佳解图，即 AO^* 具有可采纳性。当 $h(n) = 0$ 时， AO^* 也蜕化为宽度优先算法。

这里单调限制条件是指：对隐含图中，从节点 $n \rightarrow \{n_1, \dots, n_r\}$ 的每一个连接符都施加限制，即假定 $h(n) \leq C + h(n_1) + \dots + h(n_r)$ 其中 C 是连接符的耗散值。

如果 $h(n)$ 满足单调限制，且 $h(t_i) = 0 (t_i \in N)$ ，那么单调限制意味着 h 是 h^* 的下界范围，即对所有节点 n 有 $h(n) \leq h^*(n)$ 。

综上所述，看出 AO^* 与 A 有类似之处，但也有若干区别。首先是 AO^* 中评价函数只考虑 $h(n)$ 分量，因为算法有自底向上耗散值的修正操作，因而实际上局部解图耗散值的估计是在 s 处比较，即启发式估计都回溯到初始节点开始，获得应有的估计效果，计算 g 没有必要也不可能。其次是由于 k -连接符连接的有关子节点，对父节点能解与否以及耗散值都有影响，因而显然不能象 A 算法那样优先扩展其中具有最小耗散值的节点。再一点是 AO^* 算法仅适用于无环图的假设，否则耗散值递归计算不能收敛，因而在算法中还必须检查新生成的节点已在图中时，是否是正被扩展节点的先辈节点。最后还必须注意到 A 算法设有 OPEN 和 CLOSED 表，而 AO^* 算法只用一个结构 G ，它代表到目前为止已明显生成的部分搜索图，图中每一个节点的 $h(n)$ 值是估计最佳解图，而不是估计解路径。

有关 AO^* 算法的具体应用及一些提高搜索效率的修正算法，可进一步参阅人工智能有关文献。

§ 3.3 博弈树的搜索

1. 概述

博弈一向被认为是富有智能行为的游戏，因而很早就受到人工智能界的重视，早在60年代就已经出现若干博弈程序，并达到一定水平。博弈问题的研究还不断提出一些新的研究课题，从而也推动了人工智能研究的发展。

对于单人博弈的一些问题，可用一般的搜索技术进行求解，本节着重讨论双人完备信息这一类博弈问题的搜索策略。由于双人博弈可用与或树（或图）来描述，因而搜索就是寻找最佳解树（或图）的问题。

所谓双人完备信息，就是两位选手对垒，轮流走步，这时每一方不仅都知道对方过去已经走过的棋步，而且还能估计出对方未来可能的走步。对弈的结果是一方赢（另一方则输），或者双方和局。这类博弈的实例有：一字棋、余一棋、西洋跳棋、国际象棋、中国象棋、围棋等。对于带机遇性的任何博弈，因不具有完备信息，不属这里讨论范围，但有些论述可推广到某些机遇博弈中应用。

博弈问题可以用产生式系统的形式来描述，例如中国象棋，综合数据库可规定为棋盘上棋子各种位置布局的一种描述，产生式规则是各类棋子合法走步的描述，目标则可规定为将（帅）被吃掉，规则作用于数据库的结果便生成出博弈图或博弈树。下面举一个简单的例子说明博弈问题可用与或图表示，并讨论搜索策略应考虑的实际问题。

2. Grundy 博弈

Grundy 博弈是一个分钱币的游戏。有一堆数目为 N 的钱币，

由两位选手轮流进行分堆，要求每个选手每次只把其中某一堆分成数目不等的两小堆。例如选手甲把 N 分成两堆后，轮到选手乙就可以挑其中一堆来分，如此进行下去直到有一位选手先无法把钱币再分成不相等的两堆时就得认输。

分钱币问题的产生式系统描述：

综合数据库：用无序数字序列 x_1, x_2, \dots, x_n 表示 n 堆钱币不同的个数，再用两个说明符号代表选手，无序数列和符号 M 组合 $(x_1, x_2, \dots, x_n, M)$ 就代表由某个选手走步的状态。

规则：if $(x_1, \dots, x_n, M) \wedge (x_i = y + z, y \neq z)$

then $(x_1, \dots, x_{i-1}, y, z, x_{i+1}, \dots, x_n, \overline{M})$

设初始状态为 $(7, \text{MIN})$ ，则该问题的状态空间图如图 3.4 所示

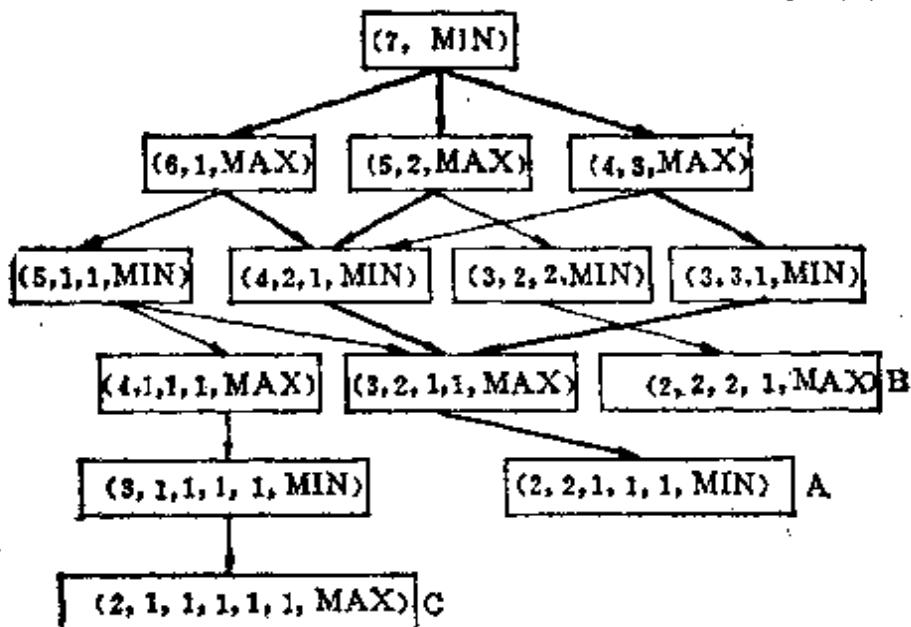


图 3.4 Grundy 博弈状态空间图

示，图中所有终节点均表示该选手必输的情况，取胜方的目标是设法使棋局发展为结束在对方走步时的终节点上，因此节点 A 是 MAX 的搜索目标，而节点 B，C 则为 MIN 的搜索目标。

搜索策略要考虑的问题是：

对 MIN 走步后的每一个 MAX 节点，必须证明 MAX 对 MIN

可能走的每一个棋局对弈后能获胜，即MAX必须考虑应付MIN的所有招法，这是一个与的含意，因此含有MAX符号的节点可看成与节点。

对MAX走步后的每一个MIN节点，只须要证明MAX有一步能走赢就可以，即MAX只要考虑能走出一步棋使MIN无法招架就成，因此含有MIN符号的节点可看成或节点。

这样一来对弈过程的搜索图就呈现出与或图表示的形式，从搜索图中可以看出，MAX存在完全取胜的策略，图中粗线所示部分就代表了这种策略，这时不论MIN怎么走，MAX均可取胜。因而寻找MAX的取胜策略便和求与或图的解图一致起来，即MAX要取胜，必须对所有与节点取胜，但只需对一个或节点取胜，这就是一个解图。因此实现一种取胜的策略就是搜索一个解图的问题，解图就代表一种完整的博奕策略。

对于Grundy这种较简单的博奕，或者复杂博奕的残局，可以用类似于与或图的搜索技术求出解图，解图代表了从开局到终局任何阶段上的弈法。显然这对许多博奕问题是不可能实现的。例如中国象棋，每个势态有40种不同的走法，如果一盘棋双方平均走50步，则搜索的位置有 $(40^2)^{50} \approx 10^{160}$ ，即深度达100层，总节点数约为 10^{161} 个。因此要考虑完整的搜索策略，就是用亿次机来处理，也得花天文数字计的时间。对于西洋跳棋、国际象棋大致也如此，而围棋更复杂了。因此即使用了强有力的启发式搜索技术，也不可能使分枝压到很少，因此这种完全取胜策略（或和局）必须丢弃，而应当把目标确定为寻找一步好棋，等对手回敬后再考虑寻找另一步好棋这种实际可行的实用策略。这种情况下每一步结束条件可根据时间限制、存储空间限制或深度限制等因素加以确定。搜索策略可采用宽度、深度或启发式方法，一个阶段搜索结束后，要从搜索树中提取一个优先考虑的“最好的”走步，这就是实用策略的基本点。下面就来讨论这种机理的

搜索策略——极小极大搜索过程。

3. 极小极大搜索过程

极小极大搜索策略是考虑双方对弈若干步之后，从可能的走步中选一步相对好棋的着法来走，即在有限的搜索深度范围内进行求解。为此要定义一个静态估计函数 f ，以便对棋局的势态（节点）作出优劣估值。这个函数可根据势态优劣特征来定义（主要用于对端节点的“价值”进行度量）。一般规定有利于 MAX 的势态， $f(p)$ 取正值，有利于 MIN 的势态， $f(p)$ 取负值，势均力敌的势态， $f(p)$ 取 0 值。若 $f(p) = +\infty$ ，则表示 MAX 赢，若 $f(p) = -\infty$ ，则表示 MIN 赢。下面的讨论规定：顶节点深度 $d=0$ ，MAX 代表程序方，MIN 代表对手方，MAX 先走。

图 3.5 是一个表示考虑两步棋的例子，图中端节点给出的数

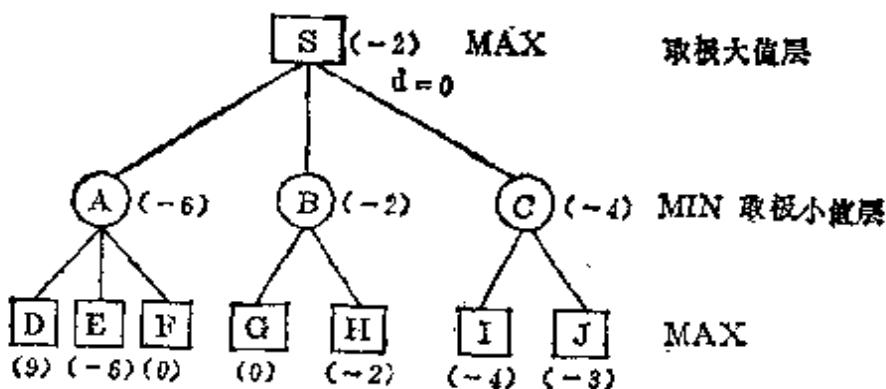


图 3.5 $f(p)$ 求值过程

字是用静态函数 $f(p)$ 计算得到，其他节点不用 $f(p)$ 估计，因为不够精确，而应用倒推的办法取值。例如 A、B、C 是 MIN 走步的节点，MAX 应考虑最坏的情况，故其估值应分别取其子节点 $f(p)$ 估值中最小者，而 S 是 MAX 走步的节点，可考虑最好的情况，故估值应取 A、B、C 值中的最大者。这样求得 $f(s) = -2$ ，

依此确定了相对较优的走步应是走向 B, 因为从 B 出发, 对手不可能产生比 $f(s) = -2$ 更差的效果。实际上可根据资源条件, 考虑更多层次的搜索过程, 从而可得到更准确的倒推值, 供 MAX 选取更正确的走步。当用端节点的静态估计函数 $f(p)$ 求倒推值时, 两位选手应采取不同的策略, 从下往上逐层交替使用极小和极大的选值方法, 故称极小极大过程。

过程 MINIMAX,

① $T := (s, \text{MAX})$, $\text{OPEN} := (s)$, $\text{CLOSED} := (\)$, 开始时树由初始节点构成, OPEN 表只含有 s。

② LOOP1, IF $\text{OPEN} = (\)$ THEN GO LOOP2;

③ $n := \text{FIRST}(\text{OPEN})$, $\text{REMOVE}(n, \text{OPEN})$, $\text{ADD}(n, \text{CLOSED})$,

④ IF n 可直接判定为赢、输或平局

THEN $f(n) := \infty \vee -\infty \vee 0$, GO LOOP1

ELSE EXPAND($n \rightarrow \{n_i\}$), $\text{ADD}(\{n_i\}, T)$

IF $d(n_i) < k$ THEN $\text{ADD}(\{n_i\}, \text{OPEN})$, GO LOOP1
ELSE 计算 $f(n_i)$, GO LOOP1, n_i 达到深度 k , 计算各端节点 f 值。

⑤ LOOP2, IF $\text{CLOSED} = \text{NIL}$ THEN GO LOOP3

ELSE $n_p := \text{FIRST}(\text{CLOSED})$;

⑥ IF ($n_p \in \text{MAX}$) \wedge ($f(n_{ci}) \in \text{MIN}$ 有值)

THEN $f(n_p) := \max\{f(n_{ci})\}$, $\text{REMOVE}(n_p, \text{CLOSED})$;

若 MAX 所有子节点均有值, 则该 MAX 取其极大值。

IF ($n_p \in \text{MIN}$) \wedge ($f(n_{ci}) \in \text{MAX}$ 有值)

THEN $f(n_p) := \min\{f(n_{ci})\}$, $\text{REMOVE}(n_p, \text{CLOSED})$;

若 MIN 所有子节点均有值, 则该 MIN 取其极小值。

⑦ GO LOOP2;

⑧ LOOP3, IF $f(s) \neq \text{NIL}$ THEN EXIT(END $\vee M(Mo-$

ve, T)); s 有值, 或则结束或标记走步。

该算法分两个阶段进行, 第一阶段②—④是用宽度优先法生成规定深度的全部博弈树, 然后对其所有端节点计算其静态估计函数值。第二阶段⑥—⑧是从底向上逐级求非端节点的倒推估计值, 直到求出初始节点的倒推值 $f(s)$ 为止, 此时

$$f(s) := \max_{i_1} \min_{i_2} \dots \{f(n_{i_1 i_2 \dots i_k})\}$$

其中 $n_{i_1 i_2 \dots i_k}$ 表示深度为 k 的端节点。再由 $f(s)$ 即可选得相对较好的走步来, 过程遂告结束。等对手响应走步后, 再以当前的状态作为起始状态 S , 重复调用该过程。下面通过最简单的 3×3 棋盘的一字棋为例来说明算法的应用过程。

在九宫格棋盘上, 两位选手轮流在棋盘上摆各自的棋子 (每次一枚), 谁先取得三子一线的结果就取胜。

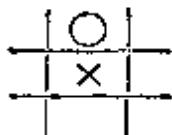
设程序方 MAX 的棋子用 (X) 表示, 对手 MIN 的棋子用 (O) 表示, MAX 先走。静态估计函数 $f(p)$ 规定如下:

若 p 对任何一方来说都不是获胜的格局, 则

$f(p) = (\text{所有空格都放上 MAX 的棋子之后, MAX 的三子成线 (行、列、对角) 的总数}) - (\text{所有空格都放上 MIN 的棋子之后, MIN 的三子成线 (行、列、对角) 的总数})$

若 p 是 MAX 获胜的格局, 则 $f(p) = \infty$; 若 p 是 MIN 获胜的格局, 则 $f(p) = -\infty$ 。

例如, 当 p 的格局如下时, 则可得 $f(p) = 6 - 4 = 2$:



设考虑走两步的搜索过程, 利用棋盘对称性的条件, 则第一次调用算法产生的搜索树如图 3.6 所示, 图中端节点下面是计算的 $f(p)$ 值, 非端节点的倒推值标记在圆圈内。为了使初始节点具

有最大的倒推值，可以看出第一步的最好着法是把棋子下在中央位置。

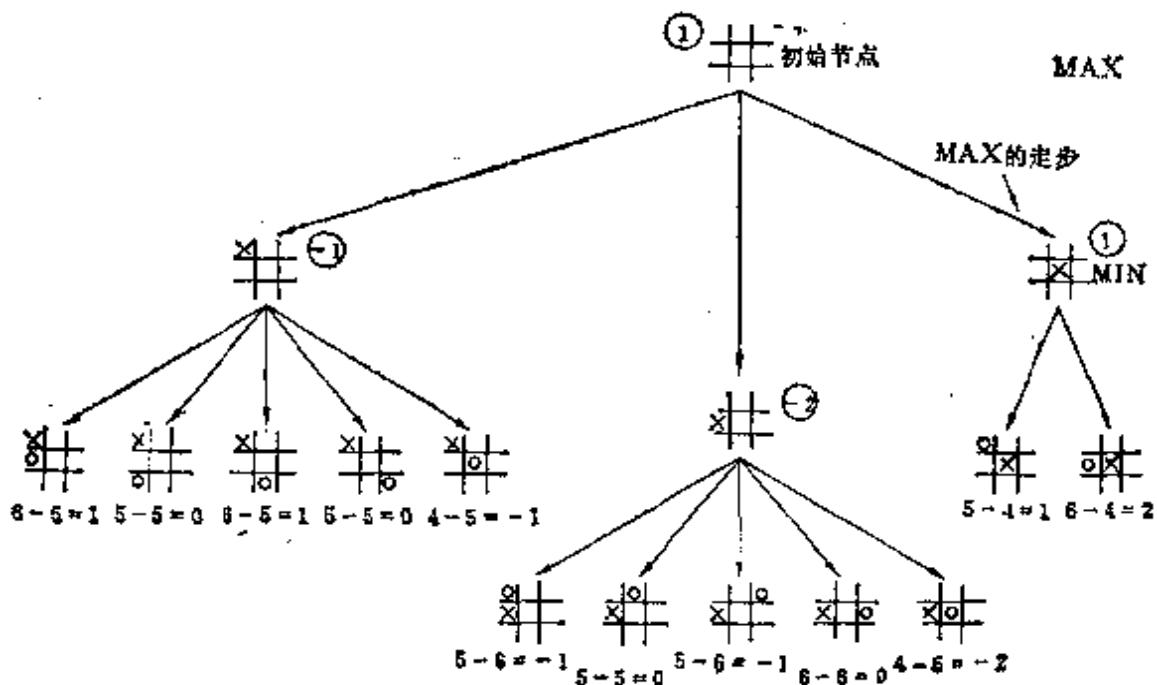


图 3.6 一字棋第一阶段搜索树

设 MAX 走完第一步后，MAX 的对手是在 \times 之上的格子下子，这时 MAX 就要在新格局下调用算法，第二次产生的搜索树如图 3.7 所示。类似的第三次的搜索树如图 3.8 所示。至此 MAX 走完最好的走步后，不论 MIN 怎么走，都无法挽回败局，因此只好认输了，否则还要进行第四轮回的搜索。

4. $\alpha-\beta$ 搜索过程

MINIMAX 过程是把搜索树的生成和格局估值这两个过程分开来进行，即先生成全部搜索树，然后再进行端节点静态估值和倒推值计算，这显然会导致低效率。如图 3.8 中，其中一个 MIN 节点要全部生成 A、B、C、D 四个节点，然后还要逐个计算其静

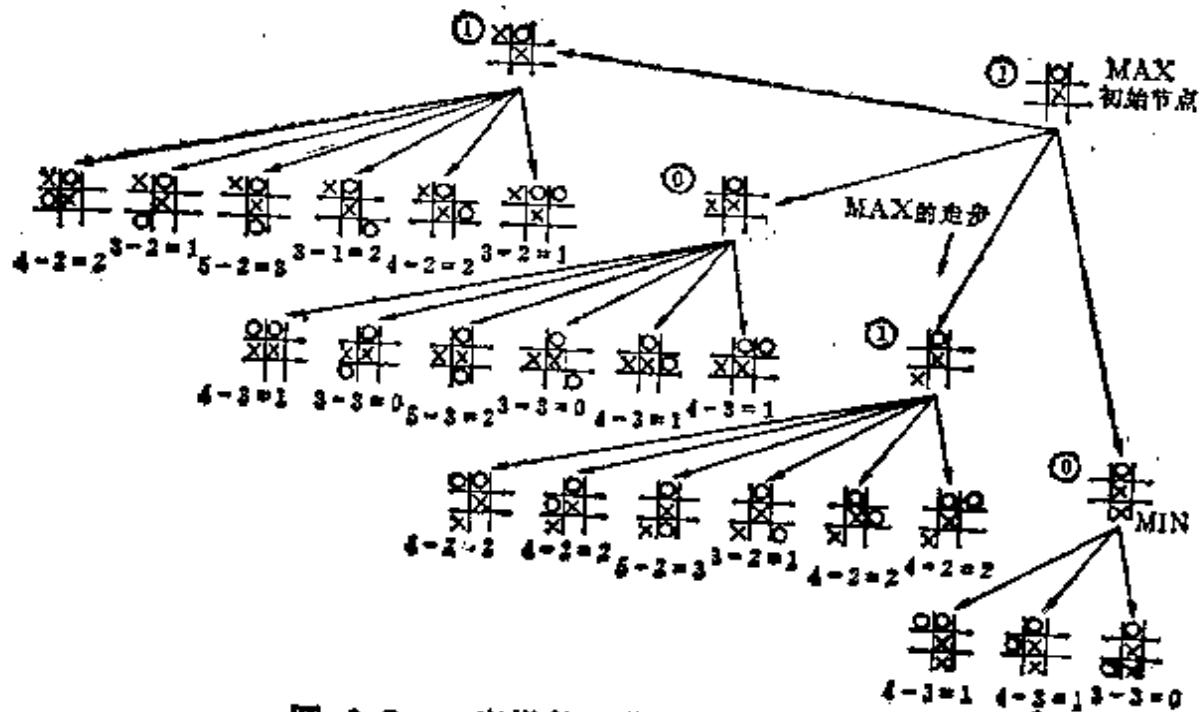


图 3.7 一字棋第二阶段搜索树

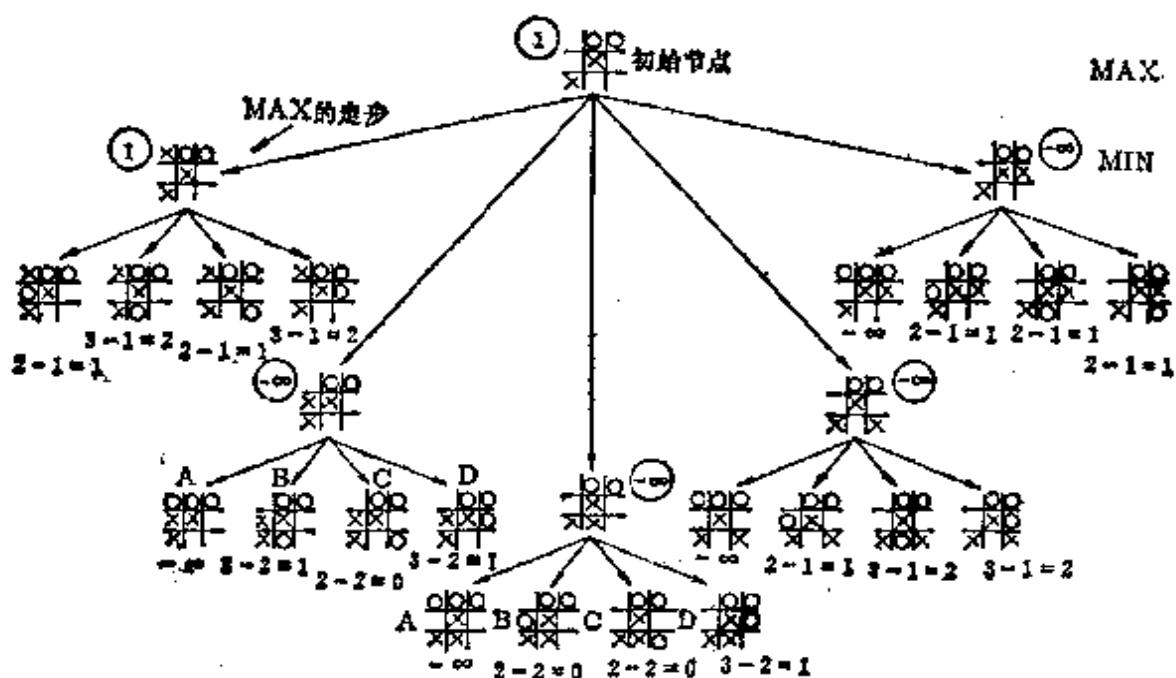


图 3.8 一字棋第三阶段搜索树

态估值，最后在求倒推值阶段，才赋给这个 MIN 节点的倒推值 $-\infty$ 。其实，如果生成节点 A 后，马上进行静态估值，得知 $f(A) = -\infty$ 之后，就可以断定再生成其余节点及进行静态计算是多余的，可以马上对 MIN 节点赋倒推值 $-\infty$ ，而丝毫不会影响 MAX 的最好优先走步的选择。这是一种极端的情况，实际上把生成和倒推估值结合起来进行，再根据一定的条件判定，有可能尽早修剪掉一些无用的分枝，同样可获得类似的效果，这就是 α - β 过程的基本思想。下面再用一字棋的例子来说明 α - β 剪枝方法。

为了使生成和估值过程紧密结合，采用有界深度优先策略进行搜索，这样当生成达到规定深度的节点时，就立即计算其静态估值函数，而一旦某个非端节点有条件确定其倒推值时就立即计算赋值。从图 3.9 中标记的节点生成顺序号（也表示节点编号）看出，生成并计算完第 6 个节点后，第 1 个节点倒推值完全确定，可立即赋给倒推值 -1 。这时对初始节点来说，虽然其他子节点尚未生成，但由于 s 属极大值层，可以推断其倒推值不会小于 -1 ，我们称极大值层的这个下界值为 α ，即可以确定 s 的 $\alpha = -1$ 。这说明 s 实际的倒推值决不会比 -1 更小，还取决于其他后继节点的倒推值，因此继续生成搜索树。当第 8 个节点生成出来并计算得静态估值为 -1 后，就可以断定第 7 个节点的倒推值不可能大于 -1 ，我们称极小值层的这个上界值为 β ，即可确定节点 7 的 $\beta = -1$ 。有了极小值层的 β 值，很容易发现若 $\alpha \geq \beta$ 时，节点 7 的其他子节点不必再生成，这不影响高一层极大值的选取，因 s 的极大值不可能比这个 β 值还小，再生成无疑是多余的，因此可以进行剪枝。这样一来，只要在搜索过程记住倒推值的上下界并进行比较，就可以实现修剪操作，称这种操作为 α 剪枝。类似的还有 β 剪枝，统称为 α - β 剪枝技术。在实际修剪过程中， α 、 β 还可以随时修正，但极大值层的倒推值下界 α 永不下降，实际的倒推值取其后继节点最终确定的倒推值中最大的一个倒推值。而极

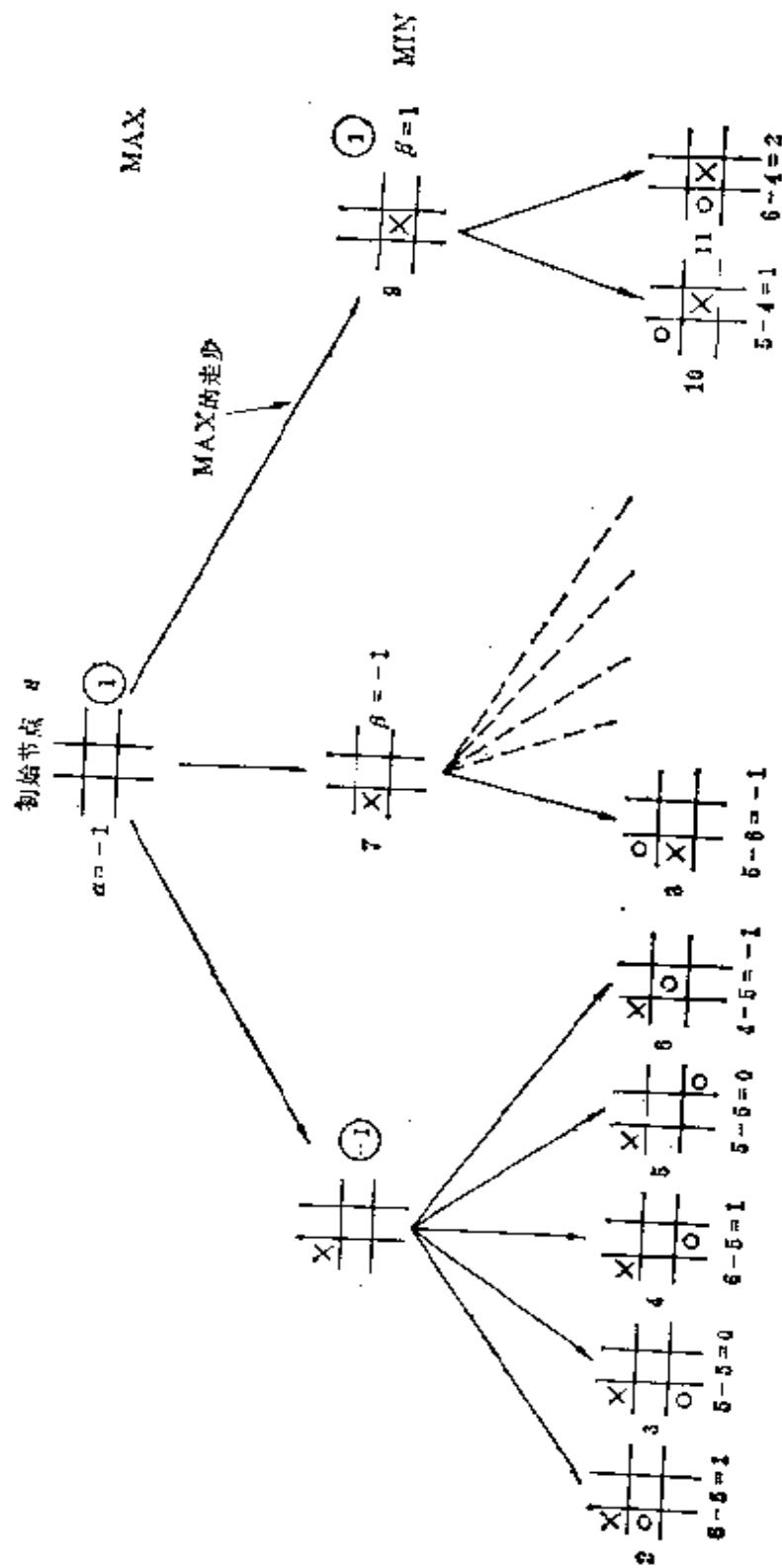


图 3.9 一子棋第一阶段 α - β 剪枝方法

小值层的倒推值上界 β 永不上升，其倒推值则取后继节点最终确定的倒推值中最小的一个倒推值。

归纳一下以上讨论，可将 α - β 过程的剪枝规则描述如下：

(1) α 剪枝：若任一极小值层节点的 β 值小于或等于它任一先辈极大值层节点的 α 值，即 α (先辈层) $\geq \beta$ (后继层)，则可中止该极小值层中这个 MIN 节点以下的搜索过程。这个 MIN 节点最终的倒推值就确定为这个 β 值。

(2) β 剪枝：若任一极大值层节点的 α 值大于或等于它任一先辈极小值层节点的 β 值，即 α (后继层) $\geq \beta$ (先辈层)，则可以中止该极大值层中这个 MAX 节点以下的搜索过程。这个 MAX 节点的最终倒推值就确定为这个 α 值。

根据这些剪枝规则，很容易给出 α - β 算法描述，显然剪枝后选得的最好优先走步，其结果与不剪枝的 MINIMAX 方法所得完全相同，因而 α - β 过程具有较高的效率。

图 3.10 给出 $d=4$ 的博弈树的 α - β 搜索过程，其中带圆圈的数字表示求静态估值及倒推值过程的次序编号。该图详细表示出 α - β 剪枝过程的若干细节。初始节点的最终倒推值为 1，该值等于某一个端节点的静态估值。最好优先走步是走向右分枝节点所代表的棋局，要注意棋局的发展并不一定要沿着通向静态值为 1 的端节点这条路径走下去，这要看对手的实际响应而定。

下面分析一下剪枝的效率问题。若以最理想的情况进行搜索，即对 MIN 节点先扩展最低估值的节点（若从左向右顺序进行，则设节点估计值从左向右递增排序），MAX 先扩展最高估值的节点（设估计值从左向右递减排序），则当搜索树深度为 D ，分枝因数为 B 时，若不使用 α - β 剪枝技术，搜索树的端节点数 $N_D = B^D$ ；若使用 α - β 剪枝技术，可以证明理想条件下生成的端节点数最少，有

$$N_D = 2B^{D/2} - 1 \quad (D \text{ 为偶数})$$

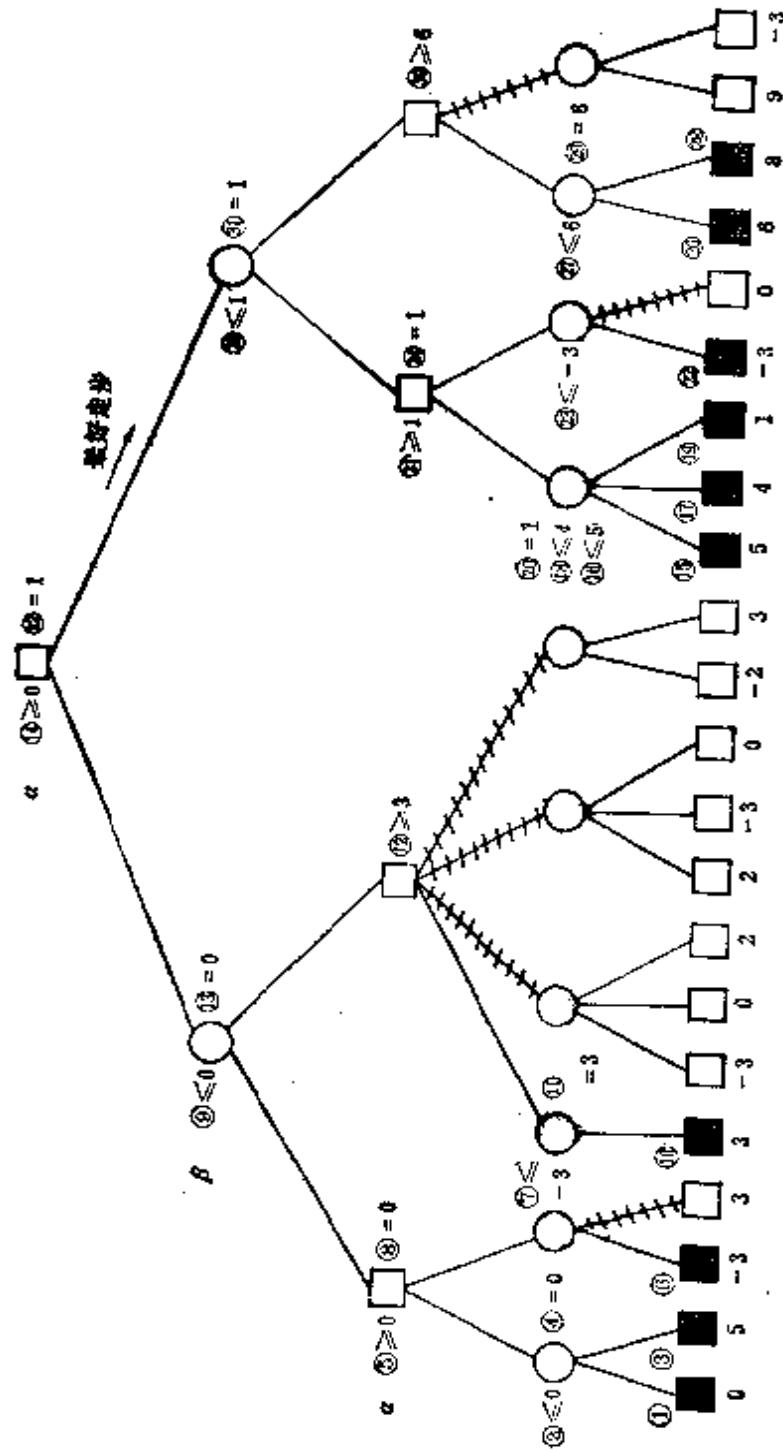


图 3.10 α - β 搜索过程的博弈树

$$N_D = B^{(D+1)/2} + B^{(D-1)/2} - 1 \quad (D \text{ 为奇数})$$

比较后得出最佳 α - β 搜索技术所生成深度为 D 处的端节点数约等于不用 α - β 搜索技术所生成深度为 $D/2$ 处的端节点数。这就是说，在一般条件下使用 α - β 搜索技术，在同样的资源限制下，可以向前考虑更多的走步数，这样选取当前的最好优先走步，将带来更大的取胜优势。

5. 其他改进方法

使用 α - β 剪枝技术，当不满足剪枝条件（即 $\alpha \geq \beta$ ）时，若 β 值比 α 值大不了多少或极相近，这时也可以进行剪枝，以便有条件把搜索集中到会带来更大效果的其他路径上，这就是中止对效益不大的一些子树的搜索，以提高搜索效率。

其他改善极小极大过程性能的基本方法有：

（1）不严格限制搜索的深度，当到达深度限制时，如出现博弈格局有可能发生较大变化时（如出现兑子格局），则应多搜索几层，使格局进入较稳定状态后再中止，这样可使倒推值计算的结果比较合理，避免考虑不充分产生的影响，这是等候状态平稳后中止搜索的方法。

（2）当 MINIMAX 法给出所选的走步后，不马上停止搜索，而是在原先估计可能的路径上再往前搜索几步，再次检验会不会出现意外，这是一种增添辅助搜索的方法。

（3）对某些博弈的开局阶段和残局阶段，往往总结有一些固定的对弈模式，因此可以利用这些知识编好走步表，以便在开局和结局时使用查表法。只是在进入中盘阶段后，再调用其他有效的搜索算法，来选择最优的走步。

以上介绍的各种博弈搜索技术可用于求解所提到的一些双人博弈问题。但是这些方法还不能全面反映人们弈棋过程实际所使用的一切推理技术，也未涉及棋局的表示和启发函数问题。例如一

些高明的棋手，对棋局的表示有独特的模式，他们往往记住的是一个可识别的模式集合，而不是单独棋子的具体位置。此外有些博弈过程，在一个短时期内短兵相接，进攻和防御的战术变化剧烈，这些情况如何能在搜索策略中加以考虑。还有基于极小极大过程的一些方法都设想对手总是走的最优走步，即我方总应考虑最坏的情况，实际上再好选手也会有失误，如何利用失误加强攻势，也值得考虑。再一点就是选手的棋风问题。总之要真正解决具体的博弈搜索技术，有许多更深入的问题需要作进一步的研究和探讨。

3.4 小 结

1. 用可分解产生式系统求解问题时，求解过程可归结为对一个隐含的与或图进行搜索。初始数据库对应于与或图的根节点，规则对应于 \wedge -连接符，结束条件的数据库对应于一组终节点集合，搜索策略的任务就是找到从初始节点到一组终节点集 N 的一个解图。解图及其耗散值可由递归定义给出。

2. 与或图的启发式搜索算法 AO^* 是通过评价函数 $f(n) = h(n)$ 来引导搜索过程，适用于分解之后得到的子问题不存在相互作用的情况。若 $S \rightarrow N$ 集存在解图，当 $h(n) \leq h^*(n)$ 且 $h(n)$ 满足单调限制条件时， AO^* 算法一定能找到最佳解图，即在这种情况下， AO^* 具有可采纳性。

3. 博弈问题可用产生式系统来描述，求解过程也是一个对与或图进行搜索的问题。本章主要讨论双人完备信息的博弈问题，一般意义下求解的策略是要找到一个完全取胜的解图。实际上只有简单的博弈或复杂博弈的残局，这种完全取胜的策略才可行。通常可行的实用策略是搜索被限制在一定的范围，搜索的目标是确定一步好棋，等对手回手后，再继续搜索。MINIMAX就

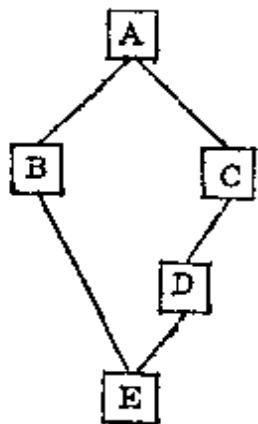
是按这种思想建立的过程，而 α - β 过程是 MINIMAX 过程的改进，并可提高效率。还有其他一些改善 MINIMAX 的方法。

习题

3.1 数字重写问题的变换规则如下：

$$\begin{array}{ll} 6 \rightarrow 3, 3 & 4 \rightarrow 3, 1 \\ 6 \rightarrow 4, 2 & 3 \rightarrow 2, 1 \\ 4 \rightarrow 2, 2 & 2 \rightarrow 1, 1 \end{array}$$

问如何用这些规则把数字6转换成一个由若干个1组成的数字串。试用算法 AO* 进行求解，并给出搜索图。求解时设 k -连接符的耗散值是 k 个单位， h 函数值规定为： $h(1) = 0$ ， $h(n) = n$ ($n \neq 1$)。

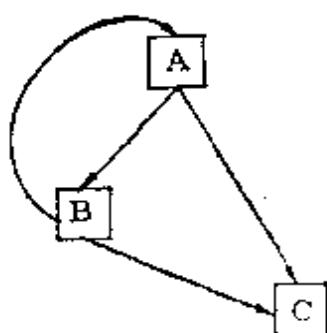


3.2 AO* 算法中，第7步从 S 中选一个节点，要求其子孙不在 S 中出现，讨论应如何实现对 S 的控制使得能有效地选出这个节点。如左图所示，若 E 的耗散值发生变化时，所提出的对 S 的处理方法应能正确工作。

3.3 如何修改 AO* 算法使之能处理出现回路的情况。如下图所示，若节点 C 的耗散值发生变化时，所修改的算法能正确处理这种情况。

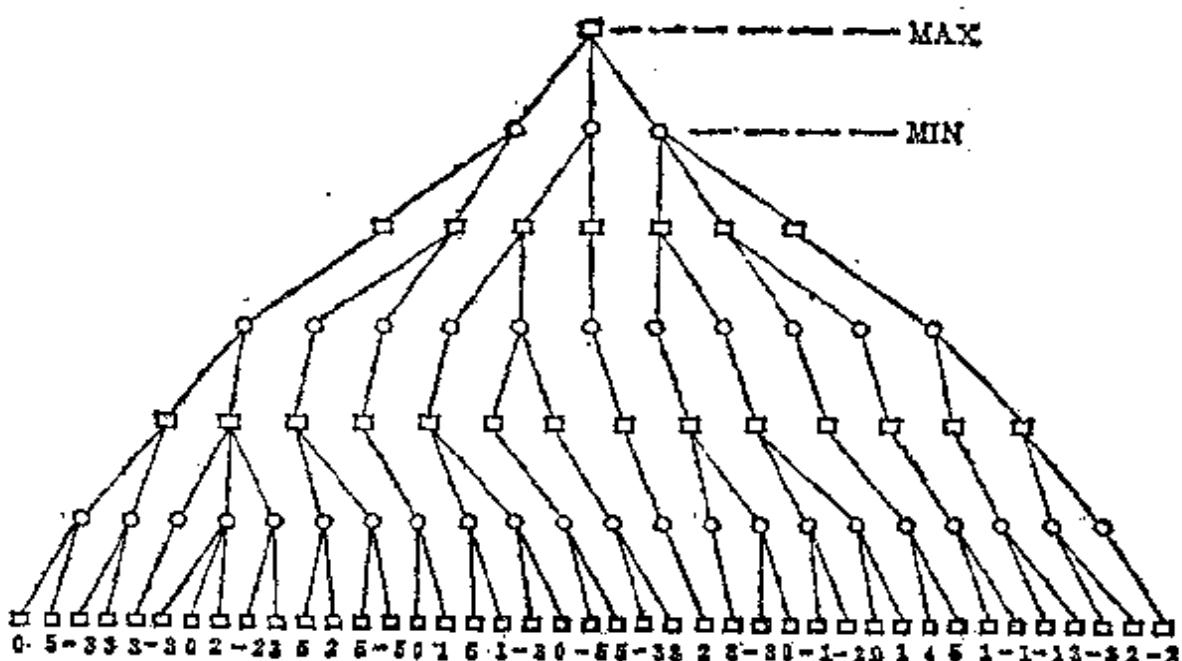
3.4 对 3×3 的一字棋，设用 +1 和 -1 分别表示两选手棋子的标记，用 0 表示空格，试给出一字棋产生式系统的描述。

3.5 余一棋的弈法如下：两棋手可以从 5 个钱币堆中轮流拿走一个、



两个或三个钱币，拣起最后一个钱币者算输。试通过博弈证明，后走的选手必胜，并给出一个简单的特征标记来表示取胜策略。

3.6 对下图所示的博弈树，以优先生成右边节点顺序来进行 α - β 搜索，试在博弈树上给出何处发生剪枝的标记，并标明属于 α 剪枝还是 β 剪枝。



3.7 写一个 α - β 搜索的算法。

3.8 用一个9维向量 C 来表示一字棋棋盘的格局，其分量根据相应格内的 \times ，空或 \circ 的标记分别用 $+1$ ， 0 ，或 -1 来表示。试规定另一个 9 维向量 W ，使得点积 $C \cdot W$ 可作为 MAX 选手（棋子标记为 \times ）估计非终端位置的一个有效的评价函数。用这个评价函数来完成几步极小-极大搜索，并分析该评价函数的效果。

第四章 人工智能中的谓词 演算及应用

谓词演算是一种形式语言，在人工智能中是一种最常用的知识表示方法，可表示各种描述语句，例如在产生式系统中，用来表达综合数据库、规则集的描述等。谓词演算体系中的一些演绎推理方法，还可用来建立自动定理证明系统、问答系统、基于规则的演绎系统等。本章将简要介绍一阶谓词逻辑的一些基本概念，讨论它们在人工智能领域中的应用问题。

4.1 一阶谓词演算的基本体系

1. 概述

一阶谓词演算体系首先规定有标点符号、括号、逻辑联结词、常量符号集、变量符号集、 n 元函数符号集、 n 元谓词符号集、量词（全称量词 \forall 和存在量词 \exists ）等。并根据这些规定，定义了谓词演算的合法表达式（原子公式、合式公式 wff），表达式的演算化简方法，以便把一般化的表达式化成为标准式（合取的前束范式或析取的前束范式）来讨论。化简结果的标准式记为

$$F \triangleleft (Q_1 x_1) \cdots (Q_n x_n) M$$

其中 $(Q_1 x_1) \cdots (Q_n x_n)$ 为前束，代表各种量词的约束关系， M 称为母式，是不包含量词符号量化的合式公式范式。

谓词演算中建立有很多推理规则，可用来从已知的公式集中推出新的公式，这些导出的公式称为定理。给出定理的推理过程及所使用的推理规则序列就构成了该定理的一个证明。在证明定理的演绎过程中，经常要对量化的表达式进行匹配操作，因而需要

对项作变量置换使表达式一致起来，这个过程称作合一。应当指出这种形式逻辑的基本概念和理论也是人工智能中最重要的概念和理论之一，在人工智能的研究中很有用。

2. 标准式的化简步骤

对任一合式公式可通过以下各步化成前束范式：

(1) 消去多余的前束(量词)。这在化简过程都要随时注意到，因为可能出现母式中没有其前束中相对应的约束变元，因而这个前束是多余的，应及时消去。

(2) 消去蕴涵符号(\rightarrow 联结词)。反复使用具有 \rightarrow 联结词的等值公式，把公式中所有的 \rightarrow 都消去。

(3) 内移否定词 \sim 的辖域范围。反复使用摩根律和量词互换式，把否定词标到只作用于原子公式为止。

(4) 变量标准化。对变量作必要的换名，使每一量词只约束一个唯一的变量名。由于变量名可任意设定，因而该过程不影响合式公式的真值。

(5) 把所有量词都集中到公式左面，移动时不要改变其相对顺序。

(6) 消去存在量词。对于待消去的存在量词，若不在任何全称量词辖域之内(即它的左边没有全称量词)，则用Skolem常量替代公式中存在量词约束的变量，若受全称量词约束(即左边有全称量词)，则要用Skolem函数(即与全称量词约束变量有关的函数)替代存在量词约束的变量，然后就可消去存在量词。例如公式

$$F = (\exists x)(\exists y)(\forall z)(\exists u)(\forall v)(\exists w)(P(x, y, z, u, v, w) \wedge (Q(x, y, z, u, v, w) \vee \sim R(x, z, w)))$$

其中母式各变量中，四个存在量词所约束的变量应分别用如下所示的Skolem函数和常量替代：

$$x=a, y=b, u=f(z), w=g(z, v)$$

消去存在量词后得

$$\begin{aligned} F_1 = & (\forall z)(\forall v)(P(a, b, z, f(z), v, g(z, v)) \\ & \wedge Q(a, b, z, f(z), v, g(z, v)) \vee \sim R(a, z, g(z, v))) \end{aligned}$$

(7) 把母式化成合取范式。反复使用结合律和分配律，将母式表达成合取范式的标准型，最后得到一个 Skolem 化的前束范式 F_1 。

(8) 隐略去前束式。由于母式的变量均受量词的约束，可省略掉全称量词，但剩下的母式仍假设其变量受全称量词量化。

(9) 把母式用子句集表示。把母式中每一个合取元称为一个子句，省去合取联结词，这样就可把母式写成集合的形式表示，每一个元素就是一个子句。如上例的子句集是

$$\begin{aligned} S = & \{P(a, b, z, f(z), v, g(z, v)), \\ & Q(a, b, z, f(z), v, g(z, v)) \vee \sim R(a, z, g(z, v))\} \end{aligned}$$

(10) 子句变量标准化。将子句集中的变量作分离标准化，即对某些变量重新命名，使任意两个子句不会有相同的变量出现。由于每一个子句都对应一个不同的合取元，变量都由全称量词量化，因而实质上两个子句的变量之间不存在任何关系，变量标准化不影响公式的真值。变量标准化这个步骤很重要，这是因为在使用子句集进行证明推理过程，有时要例化某一个全称量词量化的变量（即用某一特定值替代变量），这时就可能使公式尽量保持其一般化形式，增加了应用过程的灵活性。

3. 标准式的应用问题

经上述步骤化简得到的标准式是经过 Skolem 化的前束范式，通常也称为 S- 标准形。要注意 S- 标准形不是唯一的，若把它记为 F_S ，则 F_S 仅仅是 F （未 Skolem 化的前束式）的一个特例，取用不同的 Skolem 函数会得到不同的结果。当 F 为非永假公

式时, F_s 与 F 并不等价, 但当 F 为永假时, F_s 也一定是永假的, 即 Skolem 化并不影响 F 的永假特性。这个结论很重要, 可用定理形式描述如下, 它是下面将要讨论的归结原理的主要依据。

定理: 若 S 是合式公式 F 的 S -标准形之子句集, 则 F 为永假的充要条件是 S 为不可满足的。

上面已经提到过子句的概念, 它是 S -标准形母式中的合取元, 因而每一个子句是若干文字 (或基本式, 即原子公式和原子公式否定式组成集合中的任一元素) 的析取式。不含有文字的子句称为空子句, 显然空子句是一种没有任何能满足某种解释的子句, 即空子句的取值总为假, 一般简记为 \square 或 NIL。一个子句的文字中, 其变量被常量替代, 就得到一个文字的基例, 由基文字组成的子句称基子句。

子句集中所有元素 (即子句) 的合取式不为真, 则称该子句集为不可满足的。

最后还应指出谓词演算体系中还有一些很重要的概念, 如谓词演算的可判定性问题、逻辑推论、推理规则的有效性、推理规则系统的完备性等等, 对人工智能推理机制的研究都很有用。本章只从产生式的角度来讨论谓词演算的应用问题, 至于涉及更深入的基本理论可参阅数理逻辑的有关著述。

4.2 归结 (消解Resolution)

归结是一个重要的推理规则, 归结方法自 1965 年 Robinson 提出后, 使自动定理证明技术得到很大的发展。本节先简要介绍归结方法的基本思想, 然后再分别讨论命题逻辑和谓词逻辑的归结原理。

1. 归结原理

在定理证明系统中, 已知一公式集 F_1, F_2, \dots, F_n , 要证

明一个公式 W (定理) 是否成立, 即要证明 W 是公式集的逻辑推论时, 一种证明法就是要证明 $F_1 \wedge F_2 \wedge \cdots \wedge F_n \rightarrow W$ 为永真式。如果直接应用推理规则进行推导, 则由于演绎技巧等因素的影响, 给建立机器证明系统带来困难。另一种证明法是采用间接法 (反证法), 即不去证明 $F_1 \wedge F_2 \wedge \cdots \wedge F_n \rightarrow W$ 为永真, 而是去证明 $F = F_1 \wedge F_2 \wedge \cdots \wedge F_n \wedge \sim W$ 为永假, 这等价于证明 F 对应的子句集 $S = S_0 \cup \{\sim W\}$ 为不可满足的。这时如果用归结作为推理规则使用时, 就可以使机器证明大为简化。

归结原理的基本思路是设法检验扩充的子句集 S_1 是否含有空子句, 若 S 集中存在空子句, 则表明 S 为不可满足的。若没有空子句, 则就进一步用归结法从 S 导出 S_1 , 然后再检验 S_1 是否有空子句。可以证明用归结法导出的扩大子句集 S_1 , 其不可满足性是不变的, 所以若 S_1 中有空子句, 也就证得了 S 的不可满足性。归结过程可以一直进行下去, 这就是要通过归结过程演绎出 S 的不可满足性来, 从而使定理得到证明。下面就来讨论归结过程是怎样进行的。

2. 命题逻辑的归结原理

(1) 归结式(或预解式)的定义及性质

归结式: 对任意两个子句 C_1 和 C_2 , 若 C_1 中有一个文字 L_1 , 而 C_2 中有一个与 L_1 成互补的文字 L_2 , 则分别从 C_1 、 C_2 中删去 L_1 和 L_2 , 并将其剩余部分组成新的析取式, 则称这个新子句为归结式或预解式。

例: 设两个子句 $C_1 = L \vee C_1'$, $C_2 = (\sim L) \vee C_2'$, 则归结式 $C = C_1' \vee C_2'$ 。当 $C_1' = C_2' = \square$ 时, $C = \square$ 。

定理: 二个子句 C_1 和 C_2 的归结式 C 是 C_1 和 C_2 的逻辑推论。

证: 设 $C_1 = L \vee C_1'$, $C_2 = (\sim L) \vee C_2'$ 关于解释 I 为真, 则需证明 $C = C_1' \vee C_2'$ 关于解释 I 也为真。

关于解释 I , L 和 $\sim L$ 二者中必有一个为假。若 L 为假，则 C_1' 必为真，否则 C_1 为假，这与前提假设矛盾，所以只能是 C_1' 为真。

同理，若 $\sim L$ 为假，则 C_2' 必为真。最后有 $C = C_1' \vee C_2'$ 关于解释 I 为真，即 C 是 C_1 和 C_2 的逻辑推论。〔证毕〕

推论：子句集 $S = \{C_1, C_2, \dots, C_n\}$ 与子句集 $S_1 = \{C, C_1, C_2, \dots, C_n\}$ 的不可满足性是等价的 (S_1 中 C 是 C_1 和 C_2 的归结式，即 S_1 是对 S 应用归结法后导出的子句集)。

证：设 S 是不可满足的，则 C_1, C_2, \dots, C_n 中必有一为假，因而 S_1 必为不可满足的。

设 S_1 是不可满足的，则对于不满足 S_1 的任一解释 I ，可能有两种情况：

① I 使 C 为真，则 C_1, C_2, \dots, C_n 中必有一子句为假，因而 S 是不可满足的。

② I 使 C 为假，则根据定理有 $C_1 \wedge C_2$ 为假，即 I 或使 C_1 为假，或使 C_2 为假，因而 S 也是不可满足的。

由此可见 S 和 S_1 的不可满足性是等价的。〔证毕〕

同理可证 S_i 和 S_{i+1} (由 S_i 导出的扩大的子句集) 的不可满足性也是等价的，其中 $i=1, 2, \dots$ 。归结原理就是从子句集 S 出发，应用归结推理规则导出子句集 S_1 ，再从 S_1 出发导出 S_2 ，依此类推，直到某一个子句集 S_n 出现空子句为止。根据不可满足性等价原理，已知若 S_n 为不可满足的，则可逆向依次推得 S 必为不可满足的。由此可以看出，用归结法证明定理，过程比较简单，只涉及归结推理规则的应用问题，因而便于实现机器证明。

(2) 命题逻辑的归结过程

命题逻辑中，若给定公理集 F 和命题 P ，则归结证明过程可归纳如下：

① 把 F 转化成子句集表示，得子句集 S_0 ；

② 把命题 P 的否定式 $\neg P$ 也转化成子句集表示，并将其加到 S_0 中，得 $S = S_0 \cup \{\neg P\}$ ；

③ 对子句集 S 反复应用归结推理规则, 直至导出含有空子句的扩大子句集为止。即出现归结式为空子句的情况时, 表明已找到了矛盾, 证明过程结束。

例：设已知公理集为

.....(1)

$$(P \wedge Q) \rightarrow R \dots \dots (2)$$

$$(S \vee T) \rightarrow O \dots \dots (3)$$

Temperature (4)

求证 R 。

化成子句集表示后得

$$S = \{P, \sim P \vee \sim Q \vee R, \sim S \vee O, \sim T \vee O, T, \sim R\}$$

归结过程很简单, 可用图4.1的演绎树表示。由于根部出现空子

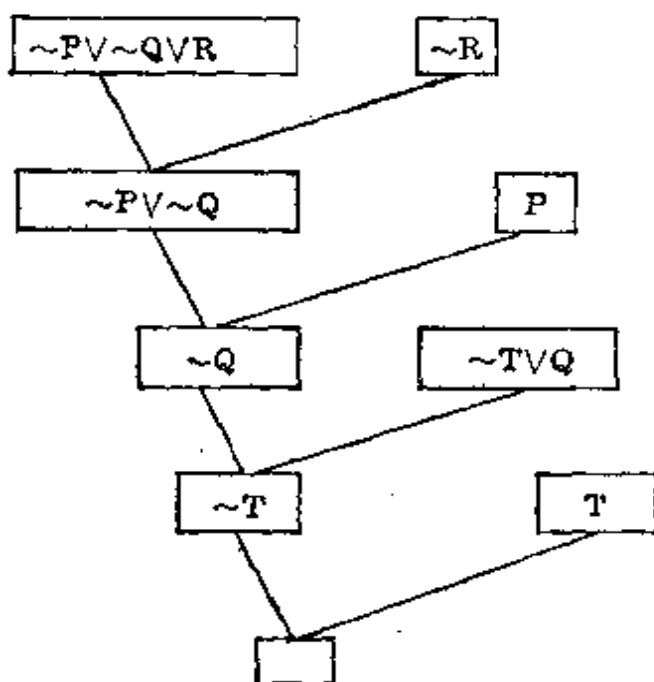


图 4.1 命题逻辑的归结演绎树

句, 因此命题R得到证明。

这个归结法证明过程其意义可解释如下：一开始是假设 S 集中所有的子句均为真，而生成的归结式表示由其他某个子句提供的信息使 S 中原子句为真的限制条件。当某一个子句限制太多时，可能就无法保证它为真了，这时就会出现矛盾，空子句就代表矛盾的出现。

具体来说，要命题(2)为真， $\neg P$ ， $\neg Q$ 或 R 中应

有一个为真，导出第一个归结式时已假定 $\neg R$ 为真，因此命题

(2) 为真就要受到限制，只能是 $\sim P$ 或 $\sim Q$ 有一个为真。而当导第二个归结式时，已知命题 P 为真，因此命题 (2) 为真的条件进一步限制为 $\sim Q$ 为真了。同理导第三个归结式时，可得出命题 (3) 的分命题 $\sim T \vee Q$ 为真的限制是 $\sim T$ 为真，但命题 (4) 言称 T 必为真，这就导致了矛盾的出现，产生了空子句，说明无法使 S 中所有子句在某一解释下均为真。结论是因命题 (1)—(4) 均为真，所以只能 $\sim R$ 为假了。

3. 谓词逻辑的归结原理

从命题逻辑的归结过程中看出，求归结式时比较容易确定含有互补对文字的两个子句，这只要检查一下 L 和 $\sim L$ 就可以了，因而匹配过程比较简单。而在谓词逻辑中，由于要考虑变量的约束问题，这一匹配过程就复杂多了，因而在应用归结法时，往往要对公式作变量置换和合一等处理，才能得到互补的基本式，以便进行归结。因此谓词逻辑的归结过程是：

- 若 S 中两子句间有相同互补文字的谓词，但它们的项不同，则必须找出对应的不一致项；
- 进行变量置换，使它们的对应项一致；
- 求归结式看能否导出空子句。

下面进一步讨论合一和归结过程。

(1) 合一(Unify)

在谓词逻辑的归结过程中，寻找项之间合适的变量置换使表达式一致是一个很重要的过程，这个过程称为合一。下面先讨论置换的几个概念，然后再给出合一算法。

一个表达式的项可以是常量符号、变量符号或函数式（由函数符号及其项组成）。表达式的例 (instance) 是指在表达式中用置换项置换变量后而得到的一个特定的表达式。例如表达式 $P[x, f(y), B]$ ，对应于不同的变换 s_i ，可得到不同的例：

置换

$$s_1 = \{z/x, w/y\}$$

$$s_2 = \{A/y\}$$

$$s_3 = \{g(z)/x, A/y\}$$

$$s_4 = \{c/x, A/y\}$$

置换的例

$$P[x, f(y), B]s_1 = P[z, f(w), B]$$

$$P[x, f(y), B]s_2 = P[x, f(A), B]$$

$$P[x, f(y), B]s_3 = P[g(z), f(A), B]$$

$$P[x, f(y), B]s_4 = P[C, f(A), B]$$

第一个例叫做原始文字的初等变式，实际上置换后只是对变量作了换名。而第四个例称作基例，即置换后项中不再含有变量。

通常用有序对的集合 $s = \{t_1/v_1, t_2/v_2, \dots, t_n/v_n\}$ 来表示任一置换，用 s 对表达式 E 作置换后的例简记为 Es 。置换集的元素 t_i/v_i 的含义是表达式中的变量 v_i 处处以项 t_i 来替换，但不允许 v_i 用与 v_i 有关的项 $t_i(v_i)$ 作置换。

有时候要对表达式多次置换，如用 s_1, s_2 依次进行置换（即 $(Es_1)s_2$ ），这时可以将这两个置换合成为一个置换（记为 s_1s_2 ）。合成置换 s_1s_2 是由两部分的置换对组成：一部分仍是 s_1 的置换对，只是 s_1 的项被 s_2 作了置换；另一部分是 s_2 中与 s_1 变量不同的那些变量对。例如

$$s_1 = \{g(x, y)/z\}, s_2 = \{A/x, B/y, C/w, D/z\}$$

$$s_1s_2 = \{g(A, B)/z, A/x, B/y, C/w\}$$

这样的合成法可使 $(Es_1)s_2 = E(s_1s_2)$ ，这表明置换的合成是可以结合的，但很容易检验出，一般情况下置换是不可交换的，即 $s_1s_2 \neq s_2s_1$ 。

若存在一个置换 s 使得表达式集 $\{E_i\}$ 中每个元素经置换后的例有 $E_1s = E_2s = E_3s = \dots$ ，则称表达式集 $\{E_i\}$ 是可合一的，这个置换 s 称作 $\{E_i\}$ 的合一者。在归结法中主要是处理可合一的子句集，例如有子句集 $\{P(x, f(y), B), P(x, f(B), B)\}$ ，若对两个子句作置换 $s = \{A/x, B/y\}$ ，则可得 $\{P(A, f(B), B)\}$ ，因而该子句集是可合一的。

仅仅从合一的角度看， s 并不是最简单的合一者，因为还存在

另一个 $s_1 = \{B/y\}$ 的合一者，使子句集合一为 $\{P[x, f(B), B]\}$ 。因此通常还要求找的是最一般或最普通的合一者 (most general unifier) g ，记为 $\text{mgu } g$ 。

任一合一者 s 与 g 之间的关系是：存在一个置换 s' ，使得 $\{E_1\}s = \{E_1\}gs'$ 。再比较上例的两个合一者，有 $\{A/x, B/y\} = \{B/y\} \{A/x\}$ ，因此 $\text{mgu } g = \{B/y\}$ 。可见 $\text{mgu } g$ 的置换限制最少，所产生的例更一般化，这有利于归结过程的灵活使用。

下面给出 UNIFY 算法，要注意算法中可合一的表达式要用表结构来表示，如把 $P(x, f(A, y))$ 表示成 $(P \ x \ (f \ A \ y))$ 。找到的合一者是可合一表达式的 mgu ，若表达式不可合一时，则算法失败退出。

递归过程 UNIFY(E_1, E_2)

① IF ATOM(E_1) OR ATOM(E_2)

THEN 交换 E_1 和 E_2 的位置使 E_1 是一个原子（有必要时），do：，原子包括谓词符号、函数符号、常数符号、变量符号和否定符号。 E_1 不是原子，而 E_2 是原子时，则交换位置，使 E_1 是一个原子。

② begin

③ IF $E_1 = E_2$ THEN RETURN NIL；

④ IF VAR(E_1) IF CONTAIN(E_2, E_1) THEN RETURN(FAIL) ELSE RETURN ($\{E_2/E_1\}$)；

E_1 是变量且 E_2 中不含有 E_1 ，返回置换 $\{E_2/E_1\}$ 。

⑤ IF VAR(E_2) THEN RETURN($\{E_1/E_2\}$)

ELSE RETURN (FAIL)； E_2 是变量返回置换 $\{E_1/E_2\}$ 。

⑥ end

⑦ $F_1 := \text{FIRST}(E_1), T_1 := \text{TAIL}(E_1)$ ； F_1 放 E_1 第一个元素，其余元素放在 T_1 中。

- ⑧ $F_2 := \text{FIRST}(E_2)$, $T_2 := \text{TAIL}(E_2)$;
- ⑨ $s_1 := \text{UNIFY}(F_1, F_2)$; 递归调用。
- ⑩ IF $s_1 = \text{FAIL}$ THEN RETURN (FAIL);
- ⑪ $G_1 := T_1 s_1$, $G_2 := T_2 s_1$; 对未匹配部分作置换。
- ⑫ $s_2 := \text{UNIFY}(G_1, G_2)$;
- ⑬ IF $s_2 = \text{FAIL}$ THEN RETURN (FAIL);
- ⑭ RETURN ($s = s_1 s_2$); 返回 s_1 和 s_2 的合成。

利用该算法可找到可合一表达式集的最一般的合一者，下面给出几个可合一集得到的结果。

$$\begin{array}{c|c|c}
 \{E_1\} & \text{mgu } g & \{E_1\}g \\
 \{P(x), P(A)\} & \{A/x\} & P(A) \\
 \hline
 \{P(f(x), y, g(y)), & \{x/y, x/z\} & P(f(x), x, g(x)) \\
 P(f(x), z, g(x))\} & & \\
 \hline
 \{P(f(x, g(A, y)), g(A))\} & \{g(A, y)/z\} & P(f(x, g(A, y)), \\
 & P(f(x, z), z))\} & g(A, y))
 \end{array}$$

合一算法是解决两个表达式匹配的问题，两个表达式都可以含有变量，通过算法求得 mgu 并进行置换后就可以得到匹配的例。在人工智能中所谓的模式匹配是一个表达式同另一个模式表达式匹配的过程，和合一过程有类似之处，只是模式匹配过程不允许变量同时出现在两个表达式中。

(2) 谓词逻辑的归结过程

设 C_1 和 C_2 为不具有完全相同变元的两个子句，子句中的变量已标准化。我们采用文字集的形式来表示子句（即文字之间理解为析取关系），则有

$$C_1 = \{C_{1i}\} \ (i=1, 2, \dots, n), \ C_2 = \{C_{2j}\} \ (j=1, 2, \dots, m)$$

再设 $\{L_{1k}\}$ 和 $\{L_{2k}\}$ 分别为 C_1 和 C_2 的两个子集。

若 $\{L_{1k}\}$ 和 $\{\sim L_{2k}\}$ 的并集存在一个 mgu s ，则两个子句的归结式为

$$C = \{\{C_{11}\} - \{L_{1k}\}\} \cup \{\{C_{21}\} - \{L_{2k}\}\},$$

可以看出对这两个子句进行归结时，由于有多种方式选取 $\{L_{1k}\}$ 和 $\{L_{2k}\}$ ，因此归结式不是唯一的。

例： $C_1 = P(x, f(A)) \vee P(x, f(y)) \vee Q(y)$

$$C_2 = \sim P(z, f(A)) \vee \sim Q(z)$$

① 取 $L_{11} = P(x, f(A))$, $L_{21} = \sim P(z, f(A))$

存在 $s = \{z/x\}$ 使 L_{11} 和 $\sim L_{21}$ 合一，所以归结式为

$$P(z, f(y)) \vee Q(y) \vee \sim Q(z)$$

② 取 $L_{11} = P(x, f(y))$, $L_{21} = \sim P(z, f(A))$

则 $s = \{z/x, A/y\}$, 归结式为

$$Q(A) \vee \sim Q(z)$$

③ 取 $L_{11} = Q(y)$, $L_{21} = \sim Q(z)$

则 $s = \{y/z\}$, 归结式为

$$P(x, f(A)) \vee P(x, f(y)) \vee P(y, f(A))$$

这个例子说明，选择不同文字对做归结时可得到不同的归结式，但由于都是用最一般的合一者作置换，因此这些归结式仍是较一般的归结式。就是说，如果对某个文字不是用 mgu 来合一，那么得到的归结式比最一般的归结式则有较多的限制。实际上我们总是希望用最一般的归结式，以增加归结过程的灵活性。下面举一个简例说明谓词逻辑的归结过程。

例：已知：① 会朗读的人是识字的；

② 海豚都不识字；

③ 有些海豚是很机灵的。

证明：有些很机灵的东西不会朗读。

首先把问题用谓词逻辑描述如下：

已知：① $(\forall x) (R(x) \rightarrow L(x))$

② $(\forall x) (D(x) \rightarrow \sim L(x))$

③ $(\exists x) (D(x) \wedge I(x))$

求证: $(\exists x) (I(x) \wedge \sim R(x))$

再把前提条件的谓词公式进行化简, 将要证明的结论取反并化成子句集, 求得子句集:

- (1) $\sim R(x) \vee L(x)$
- (2) $\sim D(y) \vee \sim L(y)$
- (3a) $D(A)$
- (3b) $I(A)$
- (4) $\sim I(z) \vee R(z)$

从子句集求归结式, 并将它加进子句集, 连续进行直到产生空子句为止。下面的归结式序列代表一个可行的证明过程:

- (5) $R(A)$ (3b) 和 (4) 的归结式
- (6) $L(A)$ (5) 和 (1) 的归结式
- (7) $\sim D(A)$ (6) 和 (2) 的归结式
- (8) NIL (7) 和 (3a) 的归结式

4.3 归结反演系统 (Refutation)

归结反演系统是用反演(反驳)或矛盾的证明法, 使用归结推理规则建立的定理证明系统。这种证明系统是基于归结的反证法, 故称归结反演系统。它不限于数学中的应用, 其基本思想还可用在信息检索、常识性推理和自动程序设计等方面的问题。

上一节已经证明了反演法证明过程的正确性。下面来讨论该过程的产生式系统描述及其搜索策略。

1. 归结反演产生式系统的基本算法

可以把不断进行归结反演的证明过程用产生式系统进行求解, 这时产生式系统的描述及算法为:

综合数据库: 子句集

规则集合: IF $c_x \in S_i$ 和 $c_y \in S_i$ 有归结式 r_{xy} ,
THEN $S_{i+1} = S_i \cup \{r_{xy}\}$

目标条件: S_n 中出现空子句

过程 RESOLUTION

- ① CLAUSES := S ; S 为初始的基本子句集。
- ② until NIL 是 CLAUSES 的元素, do:
- ③ begin
- ④ 在 CLAUSES 中选两个不同的可归结的子句 c_i 、 c_j ,
 c_i 、 c_j 称母子句。
- ⑤ 求 c_i 、 c_j 的归结式 r_{ij} ;
- ⑥ CLAUSES := $\{r_{ij}\} \cup$ CLAUSES;
- ⑦ end;

2. 搜索策略

搜索策略的主要任务是在算法的第4步中选哪两个子句做归结, 以及在第5步决定这两个子句中对哪一个文字做归结。在介绍几个实用的策略之前, 我们先通过例子讨论该产生式系统具有可交换性的特点以及搜索策略的目标问题。

现在我们用产生式系统来求解上一节所举的例子。我们已经得到一个基本的子句集

$S = \{\sim I(z) \vee R(z), I(A), \sim R(x) \vee L(x), \sim D(y) \vee \sim L(y), D(A)\}$ 若用一般的宽度优先图搜索策略求解, 其部分搜索图如图4.2所示。由图可以看出, 该系统满足可交换系统的性质, 因此可用最简单的不可撤回的策略求解, 总能找到一个证明解。即当求解过程选用了某条无用的规则之后, 不必考虑回溯, 也不必顾及规则顺序, 只要使用了若干适当的规则之后, 就必定会找到解路径。

实际上归结过程用归结反演树表示比较简单, 图4.3给出上

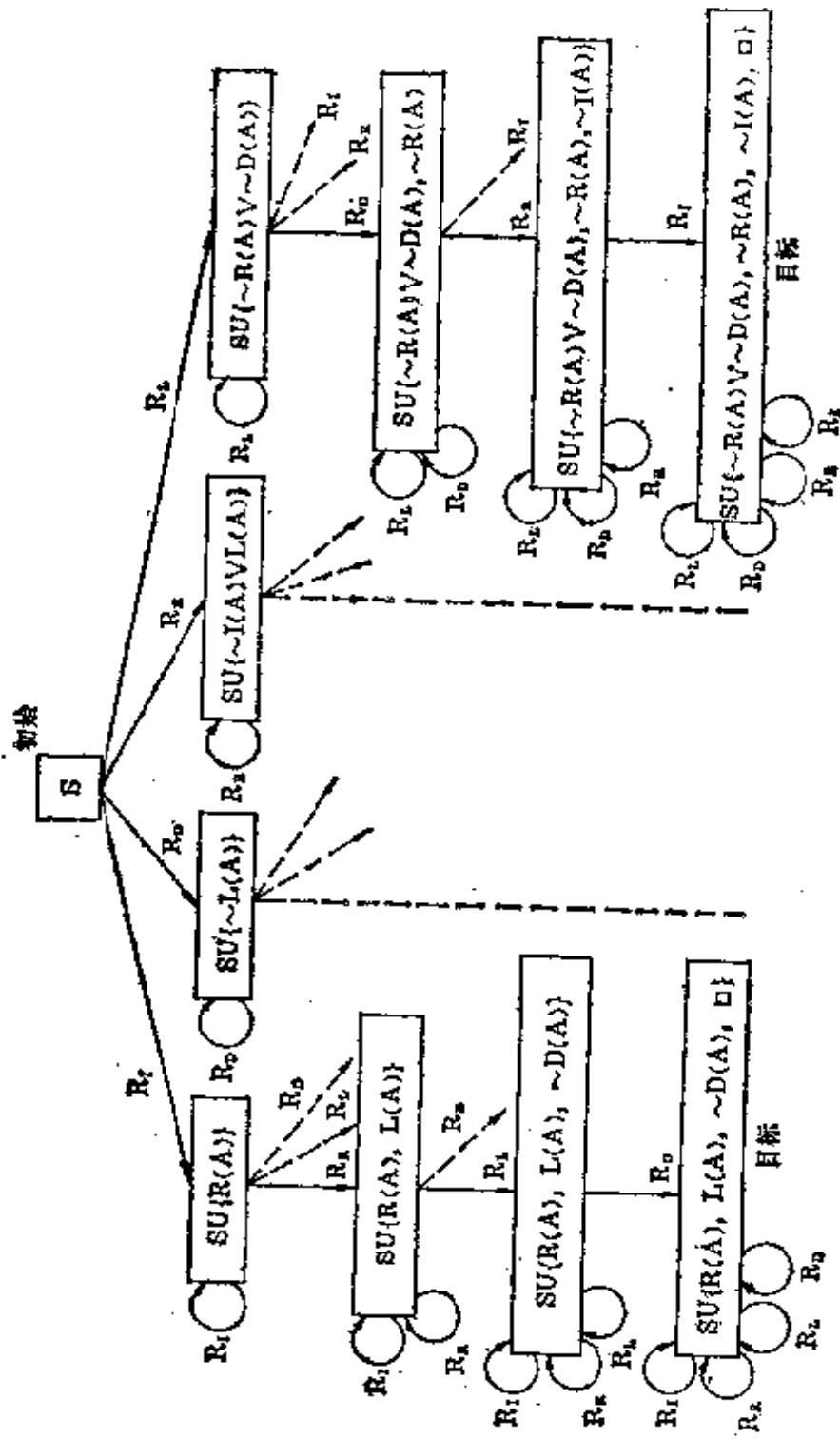


图 4.2 实例的状态空间图

例的一棵归结反演树。搜索策略的目标就是要找到一棵归结反演树。一个反演系统当存在一个矛盾时，如果使用的一种策略，最

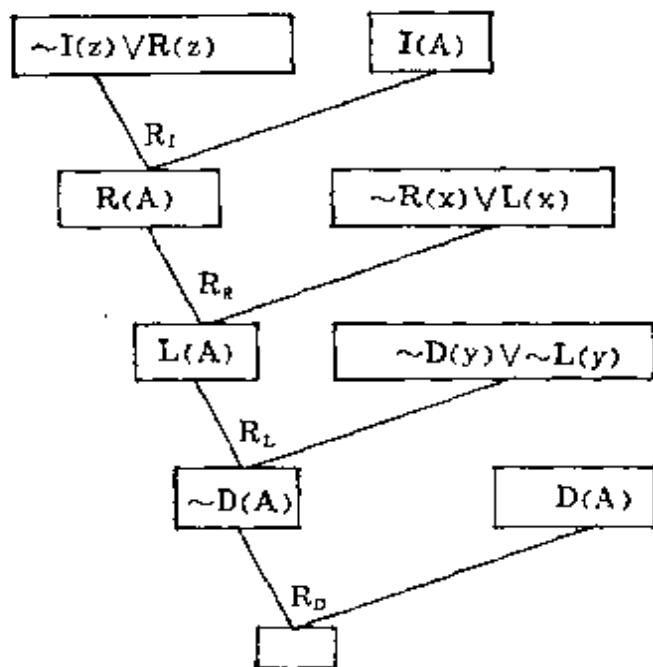


图 4.3 归结反演树

终都将找到一棵反演树（即能找到矛盾），则这种策略是完备的。在实际应用中，有些策略虽不完备，但具有极高的效率，也是可取的。提高策略效率的一些作法是：只归结含有互补对的文字；及时删去出现的重言式和被其他子句所包含的子句；每次归结都取与目标公式否定式有关的子句作为母子句之一进行归结等等。下面仅对典型的几种策略做一些简单的讨论。

(1) 宽度优先策略

宽度优先策略首先归结出基本集 S 中可能生成的所有归结式，称第一级归结式，然后生成第二级归结式等等，直到出现空子句。宽度优先策略是完备的，但效率低，图4.4给出这种策略求解上述例子的一个搜索图。

(2) 支持集策略

支持集策略是指在每一次归结时，其母子句中，至少有一个是与目标公式否定式有关的子句（即目标公式否定式本身或与该否定式有关的后裔）。可以证明支持集策略是完备的，即当矛盾存在时，一定能找到由支持集策略产生的一棵反演树。也可以把支持集策略看成是在宽度优先策略中引进某种约束条件，它代表一种启发信息，因而有较高的效率。图4.5是同一个例子的支持

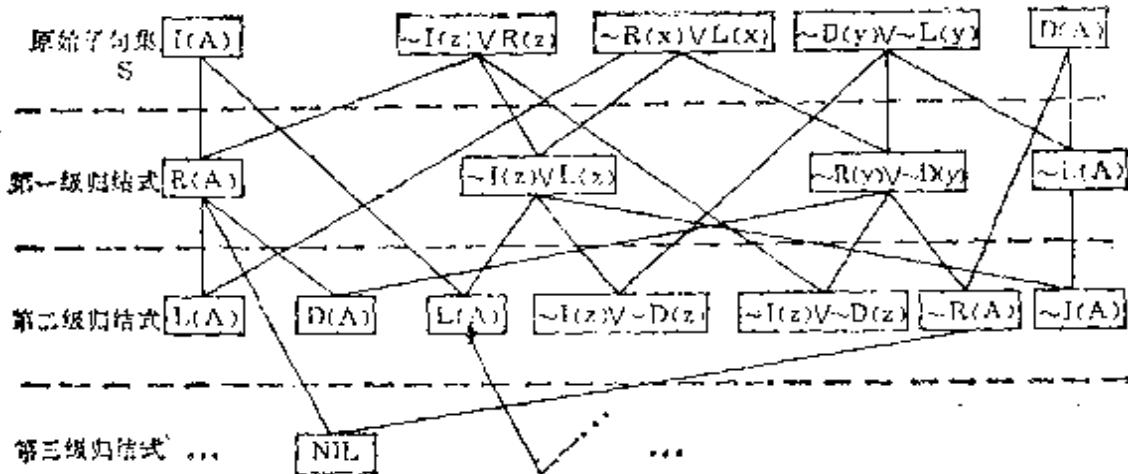


图 4.4 宽度优先搜索过程

集策略搜索图，粗线所示部分就是图4.3反演树的前三层部分。可以看出各级的归结式数目要比宽度优先策略生成的少，但在第三级还没有空子句。就是说这种策略限制了子句集元素的剧增，但会增加空子句所在的深度。此外这种策略具有逆向推理的含义，因为进行归结的一个母子句与目标子句有关，可以看成推理过程是沿目标、子目标的方向进行的。

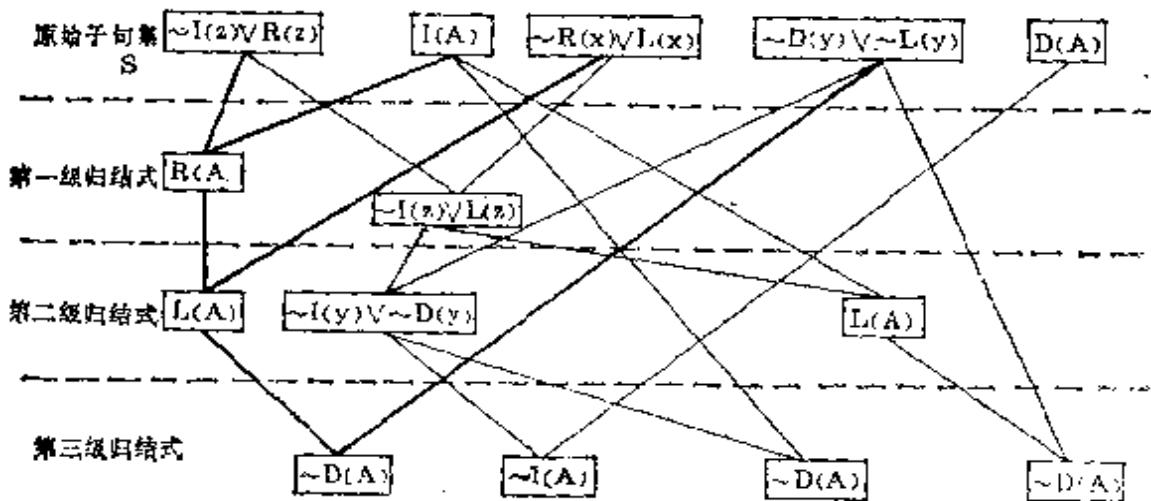


图 4.5 支持集策略搜索过程

(3) 单元子句优先策略

这种策略是支持集策略的进一步改进，每次归结时优先选取单文字的子句（称单元子句）为母子句进行归结，显然归结式的文字数会比其他情况归结的结果要少，这有利于向空子句的方向搜索，实际上会提高效率。

(4) 线性输入形策略

这种策略每次归结时，至少有一个母子句是从基本集中挑选，图4.6是这种策略的示例图。该策略可限制生成归结式的数目，具有简单和效率高的优点。但它不是一个完备的策略，我们来看一个反例：

$S = \{Q(u) \vee P(A), \neg Q(w) \vee P(w), \neg Q(x) \vee \neg P(x), Q(y) \vee \neg P(y)\}$ ，从 S 出发很容易找到一棵反演树，但不存在一个线性输入形策略的反演树。

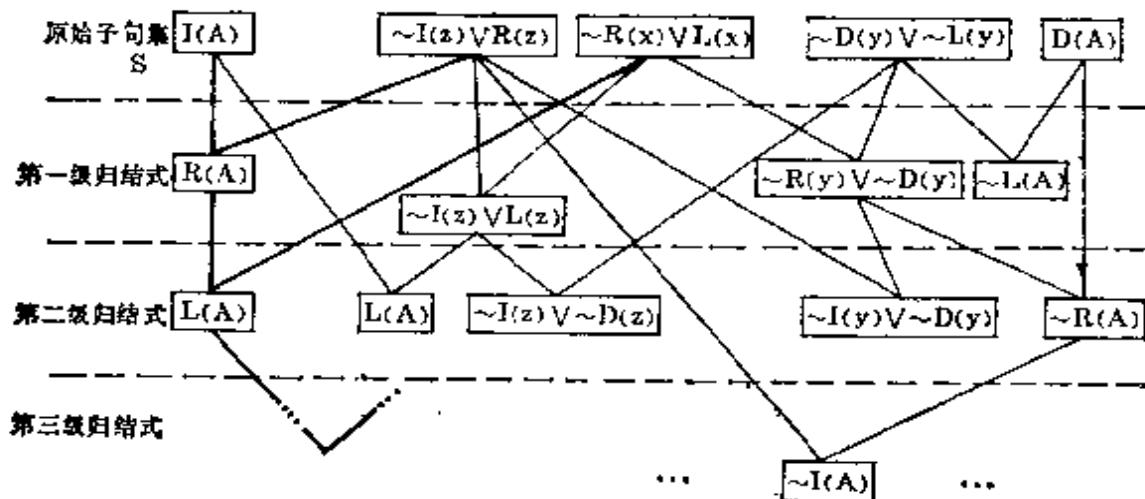


图 4.6 线性输入形策略搜索过程

(5) 祖先过滤形策略

祖先过滤形策略在每次归结时，有一个母子句或者是从基本集中挑选，或者是从另一个母子句的先辈子句中挑选，这和线性

输入形策略有点相似，但比它降低了挑选的限制。可以证明这种策略也是完备的。图4.7是这种策略求解上例的一棵反演树。

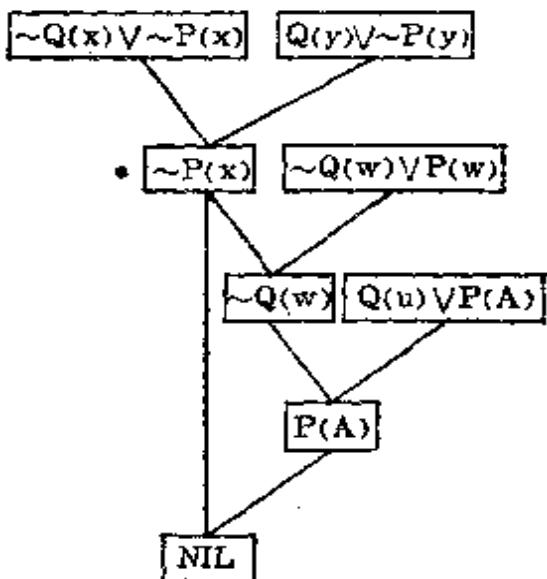


图 4.7 祖先过滤策略的搜索过程

上面分别对几种典型策略的基本思想进行了讨论，还可以采取组合的方式得到更实用的策略。总之归结反演系统搜索策略的研究主要考虑完备性和效率的问题，更全面的讨论可参阅有关文献。

4.4 基于归结法的问答系统

归结反演系统主要用来解决证明的问题，即给定公式集 F_0 ，要求证明具有存在量词量化的目标公式 $(\exists x)W(x)$ 。但有时我们希望能回答问题，即知道 x 的某一个取值或 x 的一个例，这就是问答系统的功能。对于直接回答 $x=A$ 时 $W(A)$ 是否为真这种问题，可直接用归结反演系统证明，即可给出结果。而对 $x=?$ 时 $W(x)$ 为真或填空类型的问题，则应是先证明一个与回答问题语句中提问项有对应匹配关系的一般语句，然后再找某一个具体的匹配值，该值就给出问题的回答。这种要求产生满足条件的 x 的例需要构造性的证明方法，即如果能给出对问题的构造性证明过程，那么就可能给出具体问题的回答。本节将通过具体例子讨论应用归结反演过程来提取问题的回答。

1. 提取回答的方法

例： If Fido goes wherever John goes and if John

is at school, where is Fido?

这个问题给出两个已知事实和一个询问，这个询问的答案应从事实出发演绎得到。先把问题用谓词逻辑公式表示：

前提公式集： $(\forall x)(AT(John, x) \rightarrow AT(Fido, x))$
 $AT(John, School)$

目标公式： $(\exists x)AT(Fido, x)$

我们先要证目标公式是前提公式集的逻辑推论，然后再找出一个 x 的例，这样就回答了 Fido 在何处的询问。这里主要的关键是把询问表达为一个有存在量词约束的目标公式，这样就很容易用归结法给出证明，图4.8给出了该例的反演树。接着就是如何从反演树那里对所询问的问题提取一个回答，一个简单的办法其步骤是：

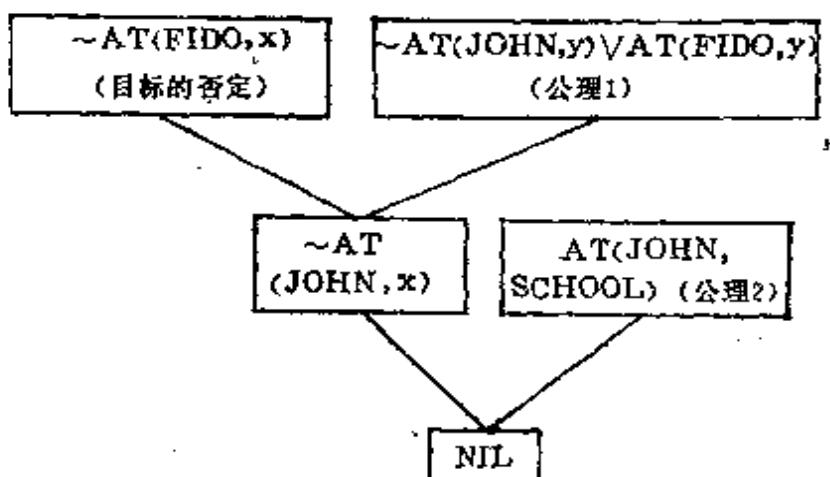


图 4.8 例题的反演树

(1) 用一个重言式来取代目标公式的否定式这个子句，该重言式为

$$\sim AT(Fido, x) \vee AT(Fido, x)$$

(2) 按反演树的构造进行归结，给出重言式替代目标否定式子句后的证明树，这时根子句不为空，称这个证明树为修改证明树，如图4.9所示。

(3) 用根部的子句作为回答语句。

从图4.9看出根部有子句 $\text{AT}(\text{Fido}, \text{School})$ 就是一个正确的回答，这个回答与目标公式形式相同，它是目标公式中约束变量 x 为常量 School 替代的结果，即求得目标公式中 x 的一个例。

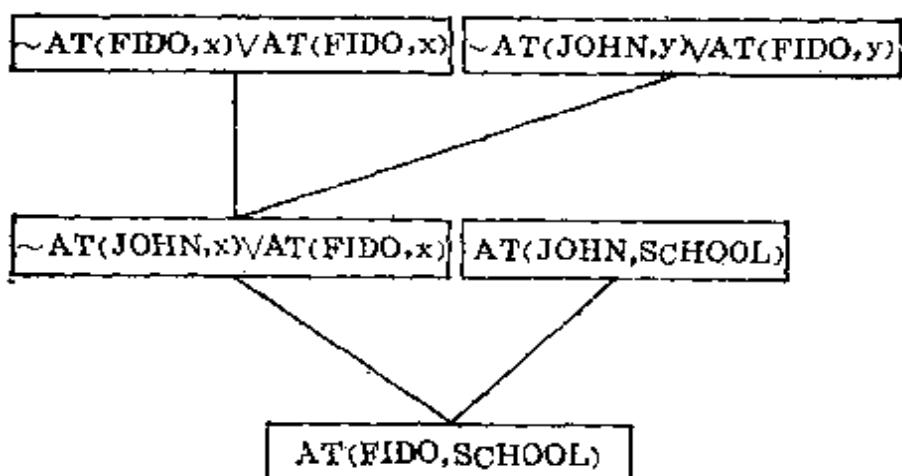


图 4.9 修改证明树

从这个例子看出，回答的提取过程是把一棵归结反演树转化为根部带有回答语句的一棵修改证明树的过程。从证明树可以得出根部的语句是公理集（前提公式集）与一个重言式（由目标否定式构成）的逻辑推论，也就是扩大公理集的逻辑推论，因而修改证明树构造的本身就证明这种提取回答的办法是正确的。

2. 提取回答的一般过程

这一节将通过几个例子，说明提取回答过程应考虑的若干具体问题，最后给出回答提取的过程。

(1) 回答中出现 Skolem 函数的情况

例1：已知：For all x and y , if x is the parent of y and y is the parent of z , then x is the grandparent of

z. Everyone has a parent.

询问: Do there exist individuals x and y such that x is the grandparent of y ?

表示成公式集后有

事实: $(\forall x)(\forall y)((P(x, y) \wedge P(y, z)) \rightarrow G(x, z))$
 $(\forall y)(\exists x)P(x, y)$

目标公式: $(\exists x)(\exists y)G(x, y)$

用归结反演法很容易就给出目标公式的证明, 其反演树如图4.10所示。图中母体子句中有横线标记的文字是归结时被合一的文字, 子句 $P(f(w), w)$ 中的 Skolem 函数 $f(w)$ 可解释为 w 父亲的名字。图4.11是这个
 问题的修改证明树, 根
 部的子句 $G(f(f(v)), v)$
 是回答语句, 形式与目
 标公式一致, 其含义由
 公式 $(\forall v)G(f(f(v));$
 $v)$ 表达, 对于每一个 v ,
 及 v 的祖父, 均是满足回
 答条件的那些个体的
 实例。

(2) 目标公式是
 析取范式的情况

这种情况比较复杂, 要规定一种方法来处理重言式中出现的
 合取符号, 才能提取出答案来。

例2: 设前提子句集是:

$\sim A(x) \vee F(x) \vee G(f(x))$
 $\sim F(x) \vee B(x)$
 $\sim F(x) \vee C(x)$

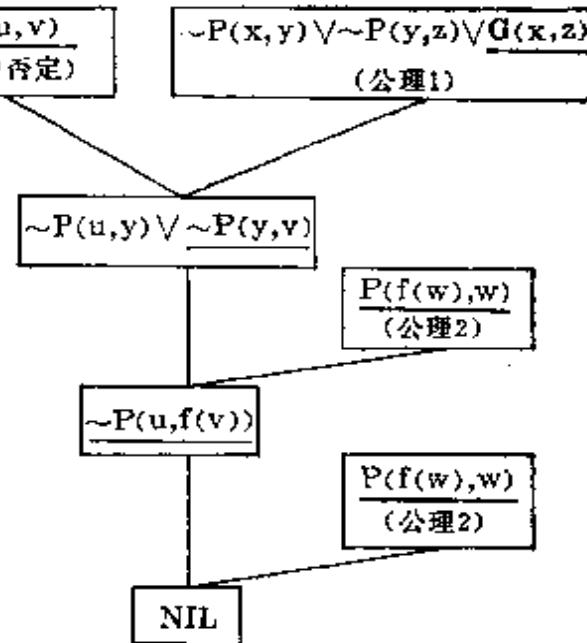


图 4.10 例1的反演树

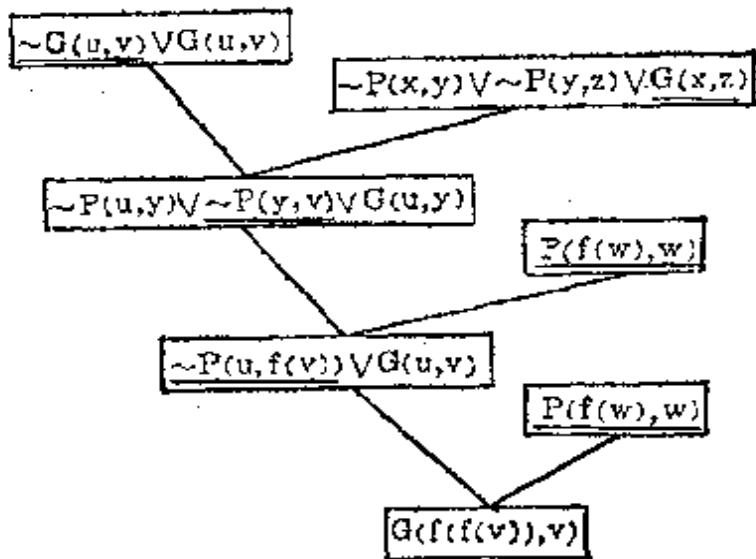


图 4.11 例1的修改证明树

$$\sim G(x) \vee B(x)$$

$$\sim G(x) \vee D(x)$$

$$A(g(x)) \vee F(h(x))$$

$$\text{目标公式: } (\exists x)(\exists y)((B(x) \wedge C(x)) \vee (D(y) \wedge B(y)))$$

先来看一下目标否定式的形式:

$$\begin{aligned} & \sim(\exists x)(\exists y)((B(x) \wedge C(x)) \vee (D(y) \wedge B(y))) \\ &= (\forall x)(\forall y)\sim((B(x) \wedge C(x)) \vee (D(y) \wedge B(y))) \\ &= (\forall x)(\forall y) ((\sim B(x) \vee \sim C(x)) \wedge (\sim D(y) \vee \sim B(y))) \end{aligned}$$

这个否定式可化为两个子句 $\sim B(x) \vee \sim C(x)$ 和 $\sim D(y) \vee \sim B(y)$ 表示, 用归结反演法对基本集 S 进行演绎, 可得反演树如图4.12所示。为了要得到修改证明树, 须根据以下两个子句来构造重言式:

$$\sim B(x) \vee \sim C(x) \vee (B(x) \wedge C(x))$$

$$\sim D(y) \vee \sim B(y) \vee (D(y) \wedge B(y))$$

这两个重言式中出现了合取符号, 不是子句表示的形式, 因此在

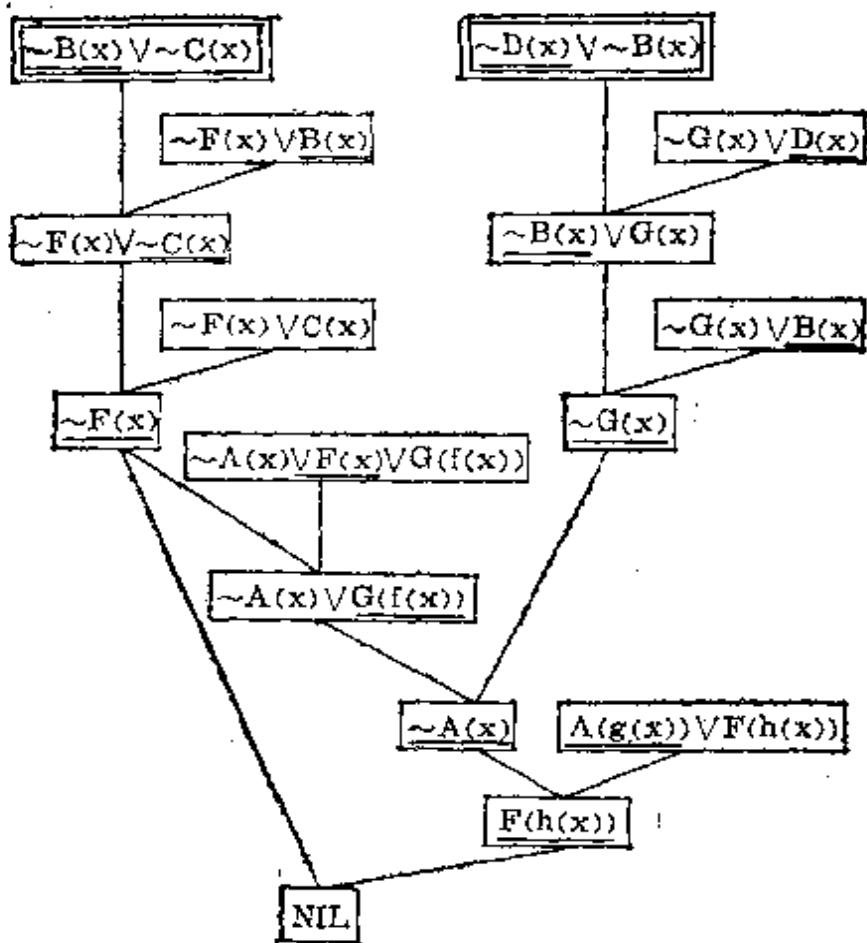


图 4.12 例2的反演树

演绎修改证明树时，要把 $(B(x) \wedge C(x))$ 和 $(D(y) \wedge B(y))$ 当作单文字式的整体来处理（因其中的元素不会取作为合一集的文字），这样便得到如图4.13所示的修改证明树，根部子句的公式是

$$(\forall x) \{ [B(g(x)) \wedge C(g(x))] \vee [D(f(g(x))) \wedge B(f(g(x)))] \vee [B(h(x)) \wedge C(h(x))] \}$$

可以看出这个回答语句的形式与目标公式略有不同，前两个析取元（画线部分）则与目标公式形式相同，第三个析取元 $[B(h(x)) \wedge C(h(x))]$ 也与目标公式的一个析取元相象，可以认为前两项已满足询问的条件。因此对于目标公式是析取范式时，提取回答

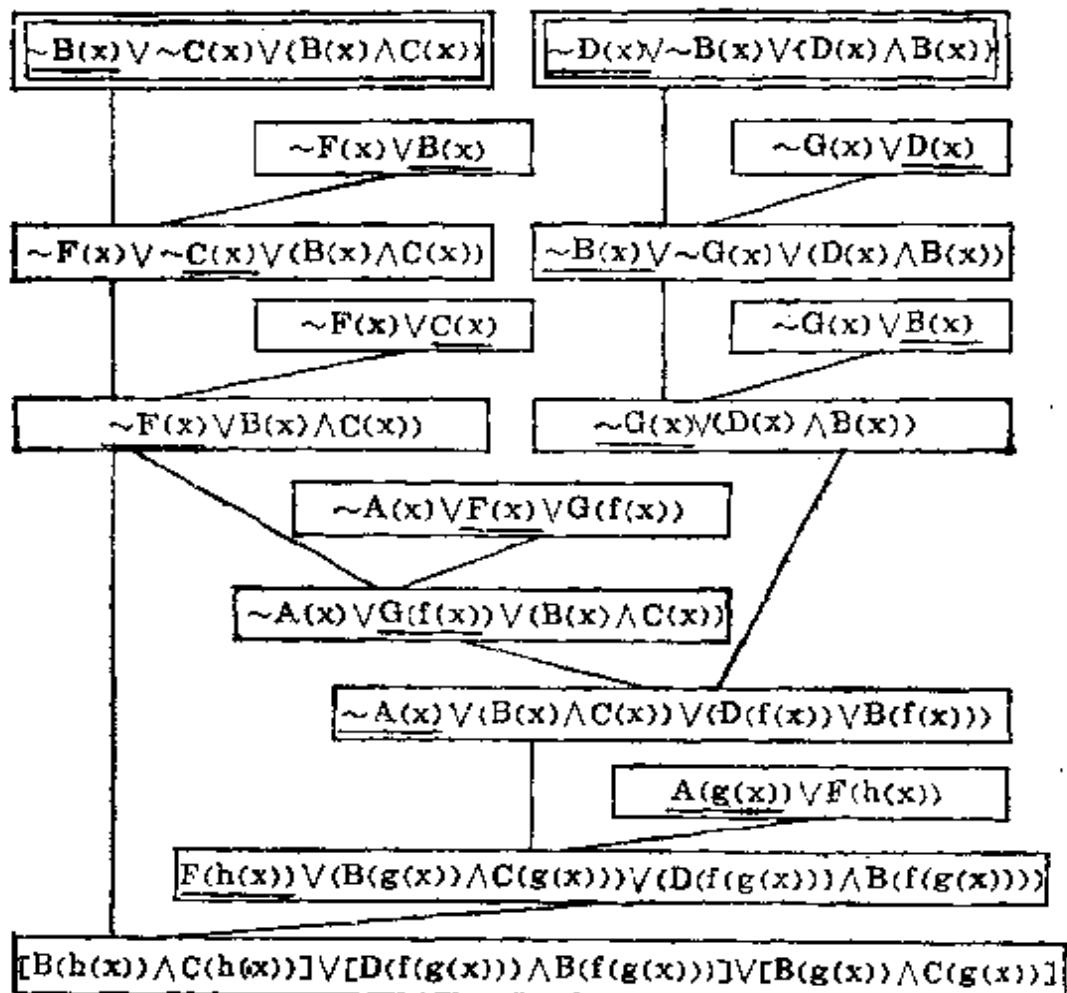


图 4.13 例2的修改证明树

过程将产生一个有类似于目标公式部分的析取项，因而也可以把根部的语句作为问题的回答。

(3) 目标公式含有全称量词量化变量的情况

在这种情况下，目标否定式出现存在量词，因而化简后目标否定式有 Skolem 函数，我们用下面的例子来说明这种情况下的处理办法。

例3：已知：For all x, x is the child of $p(x)$. For all x and y , if x is the child of y , then y is the parent of x .

询问: For any x , who is the parent of x ?

事实公式集:

$$(\forall x)C(x, p(x))$$

$$(\forall x)(\forall y)(C(x, y) \rightarrow P(y, x))$$

目标公式: $(\forall x)(\exists y)P(y, x)$

基本子句集 S 为

$$\{C(x_1, P(x_1)), \neg C(x_2, y_1) \vee P(y_1, x_2), \neg P(y_2, A)\}$$

这个例子很简单, 得到的修改证明树如图 4.14 所示, 回答语句是 $P(p(A), A)$, 其中 A 是化简目标否定式消去存在量词时引进的 Skolem 函数 (常量)。可以证明在回答提取过程中, 可以用一个新变量取代目标否定式子句中任一 Skolem 函数是正确的, 这些新变量经过替换过程仍会出现在回答的语句中, 下面再举两个例子说明这个替换过程。

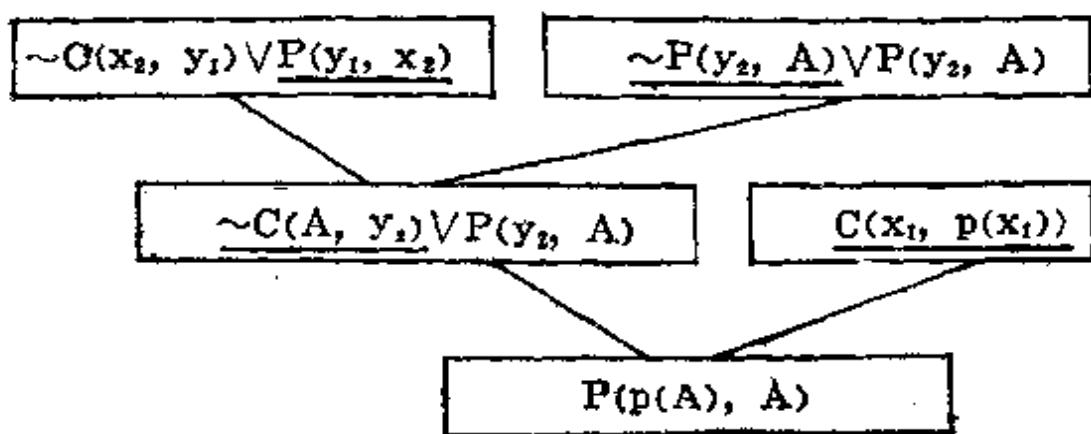


图 4.14 例 3 的修改证明树

例 4: 已知一个公理: $P(B, w, w) \vee P(A, u, u)$

目标公式: $(\exists x)(\forall z)(\exists y)P(x, z, y)$

目标否定式: $\neg P(x, g(x), y)$

它含有 Skolem 函数 $g(x)$, 如将图 4.15 反演树中 $g(x)$ 用新变量 t 替代, 则从修改证明树得到的回答语句为 $P(A, t, t) \vee P(B, z,$

z)。这个例子说明在归结一个子句时，通过换名引进的那些变量最终怎样才能出现在回答语句之中。

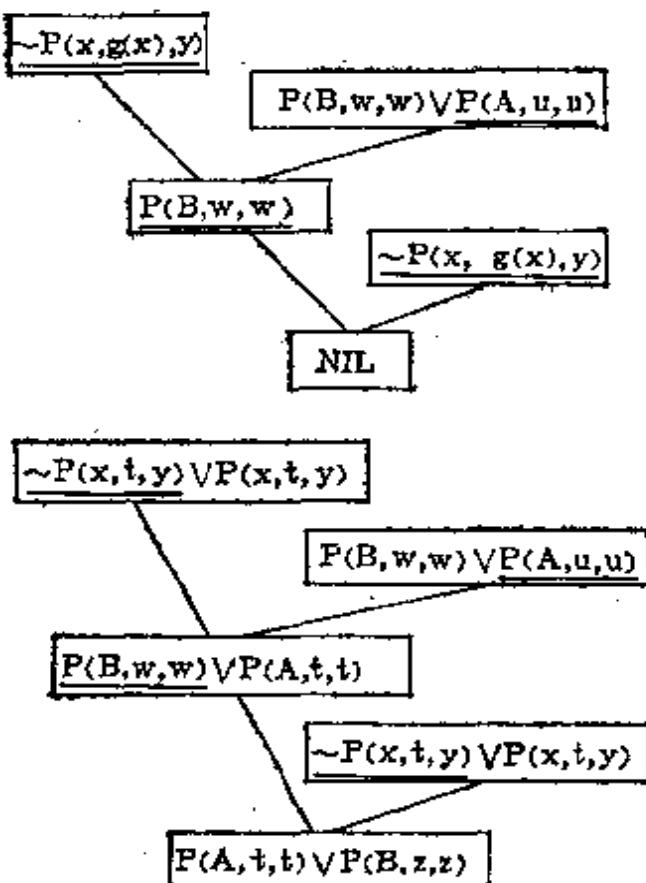


图 4.15 例 4 反演树和修改证明树

例 5：已知一个公理：

$$P(z, u, z) \vee P(A, u, u)$$

目标公式：

$$(\exists x)(\forall z)(\exists y)P(x, z, y)$$

目标否定式：

$$\sim P(x, g(x), y)$$

该例中用 w 代替 Skolem 函数 $g(x)$ ，得到的反演树和修改证明树如图 4.16 所示，可得回答语句

$$P(z, w, z) \vee P(A, w, w)$$

从这个例子看出，归结过程两棵树对应的合一集是相同的，但修改证明树中所用的置换更一般些。

通过以上几个例子，可归纳出提取回答过程的步骤如下：

(1) 使用某种搜索策略求出一个归结反演树，树中对合一集加上标记；

(2) 目标公式否定式化简得到的子句中，对出现的任一 Skolem 函数均以新变量置换；

(3) 根据目标公式否定式的子句，构造重言式；

(4) 按照已找到的归结反演树的结构，将目标否定式子句用永真式替代，且每次归结合一文字集不变，生成出修改证明树；

(5) 根部子句就作为所要提取的回答语句。

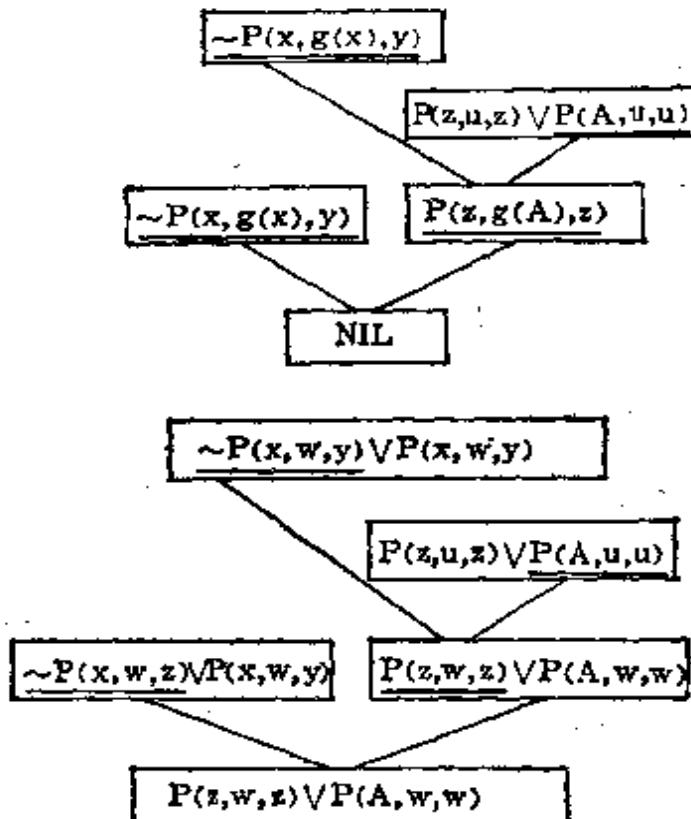


图 4.16 例 5 反演树和修改证明树

由这个过程所得到回答语句显然取决于归结反演树，因为对同一问题可能存在若干个不同的反演证明过程。这样每一个反演都可提取一个回答，而所有的回答语句中，有的可能相同，但总可能有较一般的回答语句。由于谓词演算的不可判定性，我们无法判定所提取的回答是否最一般的回答。

4.5 基于归结的自动程序综合

自动程序综合是一个困难的研究课题，目前还在发展阶段，本节只是从应用归结法的提取回答过程来讨论自动生成某些简单计算机程序的方法问题。

设要生成的程序，输入是 x ，输出是 y ， x 和 y 应满足某种特殊关系 $R(x, y)$ 。假定谓词 R 的解释适合用某个公理集来定义，而构造程序使用的基本函数的定义则由另一些公理给定。如果某个定理证明系统能证明目标公式 $(\forall x)(\exists y)R(x, y)$ 是公理集的逻辑推论，则在应用提取回答的过程中，这个证明系统就能生成出所要求的程序，即在回答语句中， y 将以若干基本函数的组合形式给出，这些函数的组合就是要生成的程序。下面举例说明这种方法的基本思想。

设我们要生成一个程序实现数字的排序，该程序的输入是一个数字表 x ，输出是另一个数字递增的有序表。描述这个程序构造的基本函数可使用 LISP 语言的系统函数 (`car`、`cdr`、`cons` 等) 和自定义的函数 (`merge`)，它们的功能定义如下：

`car(x)`：返回 x 的第一个元素；

`cdr(x)`：返回除 x 中第一个元素以外的剩余表；

`cons(e, x)`：返回元素 e 加到表 x 之前的新表，如
 $\text{cons}(\text{car}(x), \text{cdr}(x)) = x$ ；

`merge(e, l)`：返回元素 e 插入已排序表 l 中组成的一个新的有序表。

现在再引入几个公理，对问题的定义以及输入输出的关系 $R(x, y)$ 进行形式化描述：

(1) $(\forall x)(\forall y)((R(x, y) \rightarrow (S(y) \wedge I(x, y))) \wedge ((S(y) \wedge I(x, y)) \rightarrow R(x, y)))$

其中 $S(y)$ 表示表 y 是一个有序表， $I(x, y)$ 表示表 x 和表 y 元素数目相同但排序不同， $I(x, y)$ 和 $S(y)$ 可根据基本函数递归定义如下：

(2) $(\forall x)(\forall y)(\forall u)(I(x, y) \rightarrow I(\text{cons}(u, x), \text{merge}(u, y)))$

(3) $I(\text{nil}, \text{nil})$ ，(nil 是空表)

(4) $(\forall x)(\forall y)(S(y) \rightarrow S(\text{merge}(x, y)))$

(5) $S(\text{nil})$

这些公式经过化简后得子句集：

(1a) $\sim R(x, y) \vee S(y)$

(1b) $\sim R(x, y) \vee I(x, y)$

(1c) $\sim S(y) \vee I(x, y) \vee R(x, y)$

(2) $\sim I(x, y) \vee I(\text{cons}(u, x), \text{merge}(u, y))$

(3) $I(\text{nil}, \text{nil})$

(4) $\sim S(y) \vee S(\text{merge}(x, y))$

(5) $S(\text{nil})$

证明的目标公式： $(\forall x)(\exists y)R(x, y)$

我们用归纳法来证明这个目标公式，先证明长度 $n=0$ 的表，然后假设 $n \geq 0$ 成立，证明 $n+1$ 仍成立。结果得到一个对任意长度的表进行排序的递归函数。

① x 的长度 $n=0$

目标公式为 $(\exists y)R(\text{nil}, y)$ ，目标公式否定式 $\sim R(\text{nil}, y)$ ，在这种情况下，归结反演树如图 4.17 所示，由此可构造出修改证明树，从根部子句得到的回答是 $R(\text{nil}, \text{nil})$ ，其意义是若表 x 的

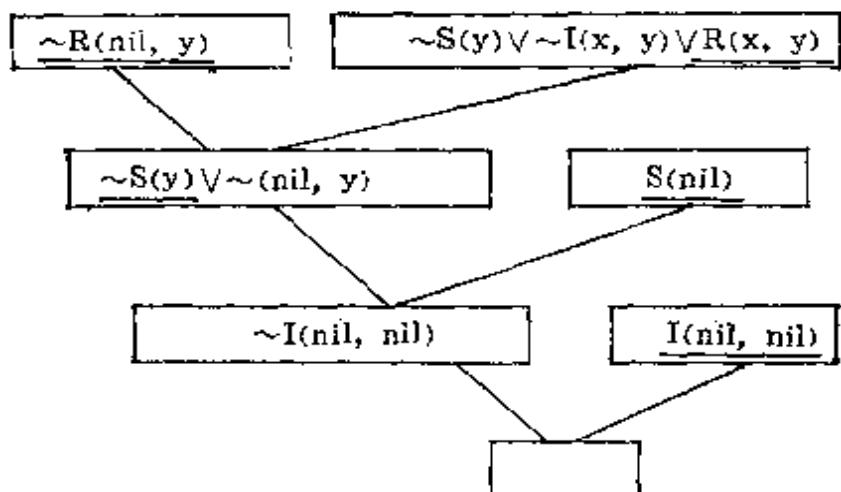


图 4.17 $n=0$ 的归结反演树

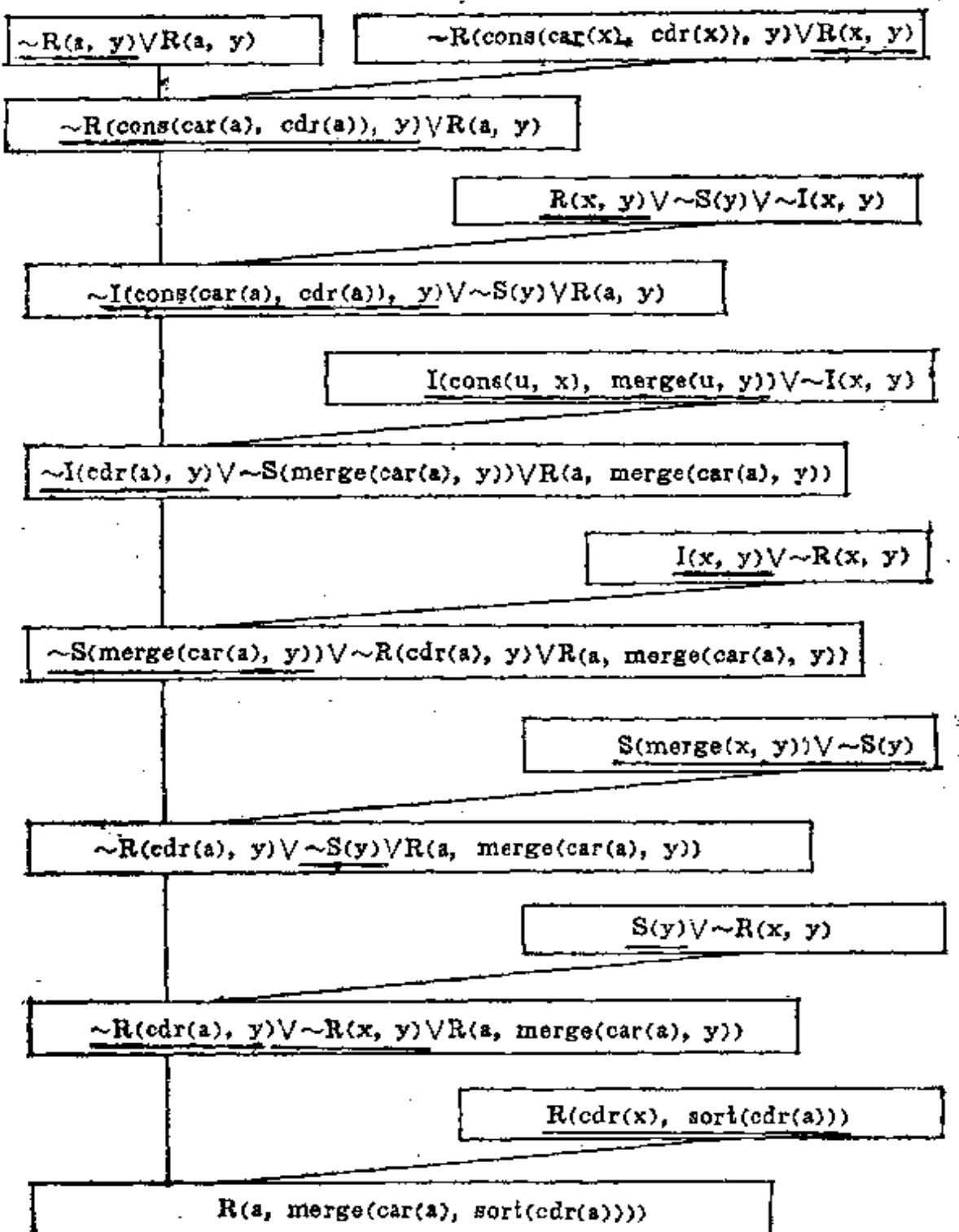


图 4.18 π 非空时的修改证明树

长度为 0，则表 y 的长度也为 0。

② x 为非空表

作归纳法假设：对任一非空表 x ， $\text{cdr}(x)$ 可以排序，即可以认为有某种排序函数“Sort”具有能够实现对小表排序的功能，这样可得归纳假设的子句形

$$(6) R(\text{cdr}(x), \text{Sort}(\text{cdr}(x)))$$

现在要证明 $(\forall x)(\exists y)R(x, y)$ ，还要用到一个关系

$$(\forall x)(\text{cons}(\text{car}(x)), \text{cdr}(x)) = x$$

为了避免处理等值谓词带来的复杂性，可引入如下事实：

$$(\forall x)(\forall y)(R(\text{cons}(\text{car}(x)), \text{cdr}(x)), y) \rightarrow R(x, y)$$

因此有

$$(7) \sim R(\text{cons}(\text{car}(x)), \text{cdr}(x)), y \vee R(x, y)$$

目标公式否定式为 $\sim R(a, y)$ ，其中 a 是 Skolem 常量，根据提取回答的过程可得修改证明树如图 4.18 所示，提取的回答语句为

$$R(a, \text{merge}(\text{car}(a), \text{sort}(\text{cdr}(a))))$$

如将 Skolem 函数用新变量替代，则可改写成

$$R(x, \text{merge}(\text{car}(x), \text{sort}(\text{cdr}(a))))$$

将以上两种情况的结果结合起来，最后可得递归程序

$$\text{Sort}(x) = \begin{cases} \text{if } x = \text{nil} \text{ then nil} \\ \text{else } \text{merge}(\text{car}(x), \text{sort}(\text{cdr}(x))) \end{cases}$$

4.6 基于归结的问题求解方法。

前面两章，我们介绍了一般产生式系统的图搜索策略和可分解产生式系统的与或图搜索策略这两种问题求解的方法，我们可以用谓词演算的合式公式对问题进行描述，并通过某种搜索策略求得问题的解。这一节将讨论通过谓词演算的定理证明系统进行

问题求解，我们用猴子摘香蕉这个经典的例子来说明这个方法。

在天花板上吊有一串香蕉的房间里，有一个可移动的箱子，问一只猴子如何规划自己的行动使得能摘到香蕉，图 4.19 给出这个问题的示意图。

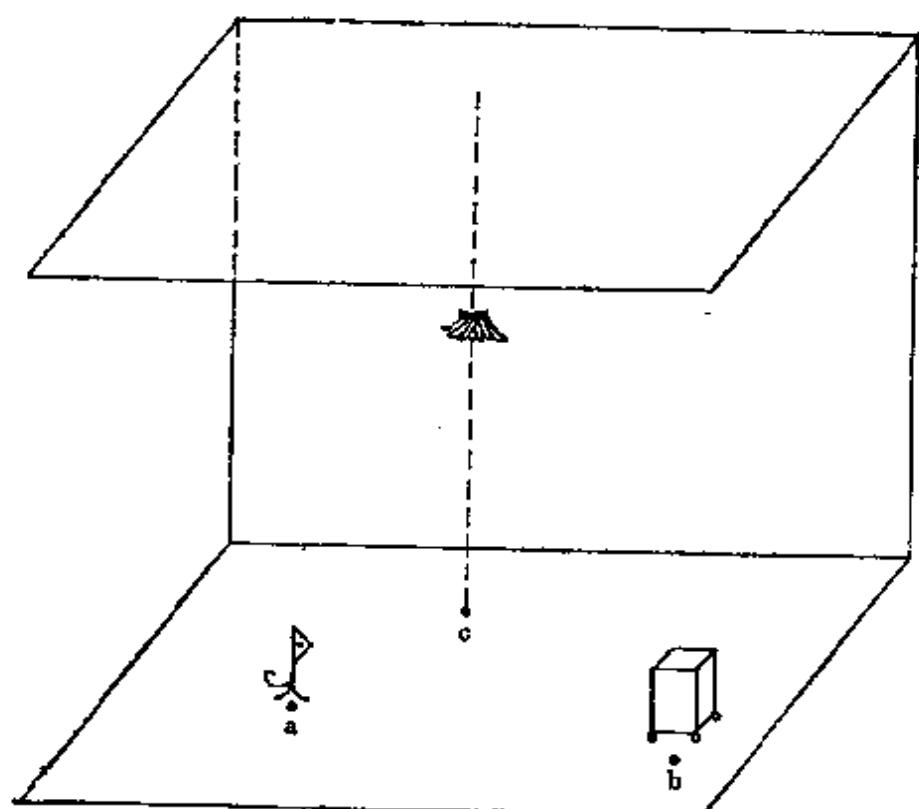


图 4.19 猴子摘香蕉问题

首先用谓词逻辑的公式对问题进行描述。初始状态可表示为

$S_0: \sim \text{ONBOX}$

$\text{AT}(\text{box}, b)$

$\text{AT}(\text{Monkey}, a)$

$\sim \text{HB}$

三个谓词的含义是：当猴子在箱顶上时， ONBOX 取真，当猴子拿到香蕉时， HB 取真；谓词 $\text{AT}(y, x)$ 当 y 处于 x 位置时取真。

目标状态为 HB。

猴子的行为可用 4 个规则（算子或操作）表示，每条规则的描述形式均用谓词演算的公式组表示，P 部分是前提条件，即规则的可应用条件，D 是规则应用后，应从状态中删去部分，A 则是添加部分。规则具体描述如下：

goto(u)

P: $\sim \text{ONBOX} \wedge (\exists x) \text{AT}(\text{monkey}, x)$

D: $\text{AT}(\text{monkey}, x)$

A: $\text{AT}(\text{monkey}, u)$

pushbox(v)

P: $\sim \text{ONBOX} \wedge (\exists x) (\text{AT}(\text{monkey}, x) \wedge \text{AT}(\text{box}, x))$

D: $\text{AT}(\text{monkey}, x)$

$\text{AT}(\text{box}, x)$

A: $\text{AT}(\text{monkey}, v)$

$\text{AT}(\text{box}, v)$

climbbox

P: $\sim \text{ONBOX} \wedge (\exists x) (\text{AT}(\text{monkey}, x) \wedge \text{AT}(\text{box}, x))$

D: $\sim \text{ONBOX}$

A: ONBOX

grasp

P: $\text{ONBOX} \wedge \text{AT}(\text{box}, c)$

D: $\sim \text{HB}$

A: HB

以这些描述为基础，就可用状态空间法或归约法进行求解。但为了能应用提取回答的过程来求解这个问题，还必须在谓词中引入状态项 s，以说明谓词应用的具体状态条件，如把初始状态的公式集表示为：

{ $\sim \text{ONBOX}(s_0), \text{AT}(\text{box}, b, s_0), \text{AT}(\text{monkey}, a, s_0),$

$\sim \text{HB}(s_0)\}$

此外对规则也必须按谓词演算的体系进行形式化描述，如对 $\text{grasp}(s)$ 的作用要用如下的合式公式表示：

$(\forall s)(\text{ONBOX}(s) \wedge \text{AT}(\text{box}, c, s) \rightarrow \text{HB}(\text{grasp}(s)))$

其含义是对所有状态 s ，若猴子在箱子上，且箱子处在 c 点处，那么在 s 状态下应用 grasp 操作的情况下，猴子就拿到了香蕉。

下面以一个简单的初始状态 $a=b$ 和只需要三条规则的情况来说明如何形式化，并应用提取回答过程进行求解的方法。这时初始状态公式集为

$\{\sim \text{ONBOX}(s_0), \text{AT}(\text{box}, b, s_0), \text{AT}(\text{monkey}, b, s_0),$
 $\sim \text{HB}(s_0)\}$

可简化为 $\sim \text{ONBOX}(s_0)$ ，于是有

- (1) $\sim \text{ONBOX}(s_0)$
- (2) $(\forall x)(\forall s)(\sim \text{ONBOX}(s) \rightarrow \text{AT}(\text{box}, x, \text{pushbox}(x, s)))$
- (3) $(\forall s)(\text{ONBOX}(\text{climbbox}(s)))$
- (4) $(\forall s)((\text{ONBOX}(s) \wedge \text{AT}(\text{box}, c, s)) \rightarrow \text{HB}(\text{grasp}(s)))$
- (5) $(\forall x)(\forall s)(\text{AT}(\text{box}, x, s) \rightarrow \text{AT}(\text{box}, x, \text{climbbox}(s)))$
- (6) $(\exists s)\text{HB}(s)$

公式组中，(1) 是初始状态，(6) 是目标公式，(2) — (4) 是规则描述公式，(5) 是推理常识，表示猴爬上箱子时，箱子的位置仍然不变，当然还可以引入其他的推理常识。

为了用归结法证明目标公式，把它们化为子句表示形式：

- (1) $\sim \text{ONBOX}(s_0)$
- (2) $\text{ONBOX}(s_1) \vee \text{AT}(\text{box}, x_1, \text{push}(x_1, s_1))$
- (3) $\text{ONBOX}(\text{climbbox}(s_2))$

(4) $\sim \text{ONBOX}(s_3) \vee \sim \text{AT}(\text{box}, c, s_3) \vee \text{HB}(\text{grasp}(s_3))$
 (5) $\sim \text{AT}(\text{box}, x_4, s_4) \vee \text{AT}(\text{box}, x_4, \text{climbbox}(s_4))$
 (6) $\sim \text{HB}(S_5)$

图 4.20 给出该问题的一棵修改证明树, 从根部得到回答语句

$\text{HB}(\text{grasp}(\text{climbbox}(\text{pushbox}(c, s_0))))$

由此可得问题的解: 猴子为拿到香蕉, 他的行动规划应是
 $(\text{push}(c, s_0), \text{climbbox}, \text{grasp})$ 。

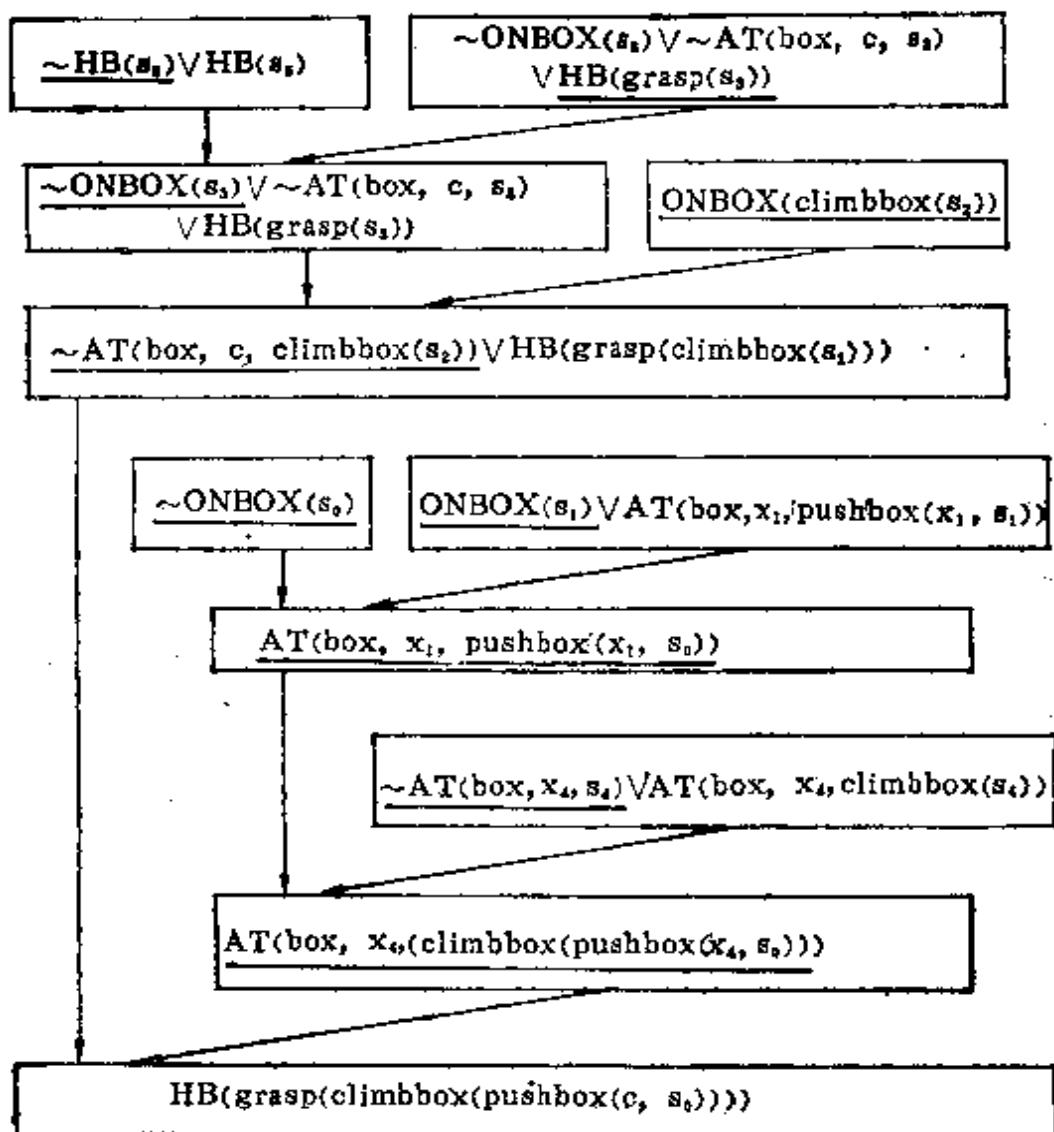


图 4.20 猴子摘香蕉问题的修改证明树

基于归结的问题求解方法，其优点是无需专门的机理来实现算子的各种计算，这些计算都是由定理证明程序中的演绎方法来处理，因此求解过程简单。其缺点是有用的启发信息不容易引入求解过程，且所需要的推理常识也都要以专门的公理形式给出各种关系的描述。

4.7 基于规则的正向演绎系统

在人工智能系统中，谓词逻辑公式常可用来表示各种知识，通常很多应用知识是用蕴涵形直接表达，因此都带有超逻辑的或启发式的控制信息。在归结反演证明系统中，要把这些表达式化成子句形表示，这就可能丢失掉包含在蕴涵形中有用的控制信息，例如下面的几个蕴涵式

$$\begin{aligned} \sim A \wedge \sim B \rightarrow C, \quad \sim A \wedge \sim C \rightarrow B, \quad \sim B \wedge \sim C \rightarrow A, \\ \sim A \rightarrow B \vee C, \quad \sim B \rightarrow A \vee C, \quad \sim C \rightarrow A \vee B \end{aligned}$$

其中任意一个式子在逻辑上都与子句 $(A \vee B \vee C)$ 等价，而每种形式表达的蕴涵式都有其内在含有的各不相同的超逻辑的控制信息，子句形表示只给出了谓词间的逻辑关系。因此有时候会希望系统能接近于原始给定的描述形式来使用这些公式，不把它们都化成子句集，这就是基于规则演绎系统的基本思想。

一般情况下，表述有关问题的知识分两类：规则和事实。规则的公式由蕴涵形给出的若干语句组成，是表示某一特定领域中的一般知识，并可以当作产生式规则来使用。事实公式则不以蕴涵形给出，是表示该问题领域的专门知识。本节讨论的演绎系统就是根据这些事实和规则来证明一个目标公式，这种定理证明系统是直接法的证明系统而不是反演系统。一个直接系统不一定比反演系统更有效，但其演绎过程容易为人们所理解。这类系统主要强调使用规则进行演绎，故称基于规则的演绎系统。

1. 事实表达式的与或形式及其表示

对一个正向演绎系统而言，事实表达式是其前提条件，是作为初始数据库描述的。这些事实化简只变换成不具蕴涵形式的与或形表示，不必完全化简为子句形，例如有事实表达式

$$(\exists u)(\forall v)(Q(v, u) \wedge \neg((R(v) \vee P(v)) \wedge S(u, v)))$$

把它化成为

$$Q(v, A) \wedge ((\neg R(v) \wedge \neg P(v)) \vee \neg S(A, v))$$

再将变量进行换名，使不同的合取元具有不同的变量名，改名后有

$$Q(w, A) \wedge ((\neg R(v) \wedge \neg P(v)) \vee \neg S(A, v))$$

这就是或与形的表示，在第二个合取元中，没有再加以展开成为范式，因此这种表示不是子句形，它与化简前的原始表达式更接近一些。

一种与或图形式可用来表示这个与或形表达式，图中每一个节点代表其中一个子表达式，子表达式间的与、或关系规定如下：

当母表达式为 $E_1 \vee E_2 \vee \dots \vee E_k$ 时，则每一个子表达式 E_i 被表示成一个后继节点，并由一个 k -连接符来连接，即表示成与关系；

当母表达式为 $E_1 \wedge E_2 \wedge \dots \wedge E_k$ 时，则每一个子表达式 E_i 均由 1-连接符连接，即表示成或关系。

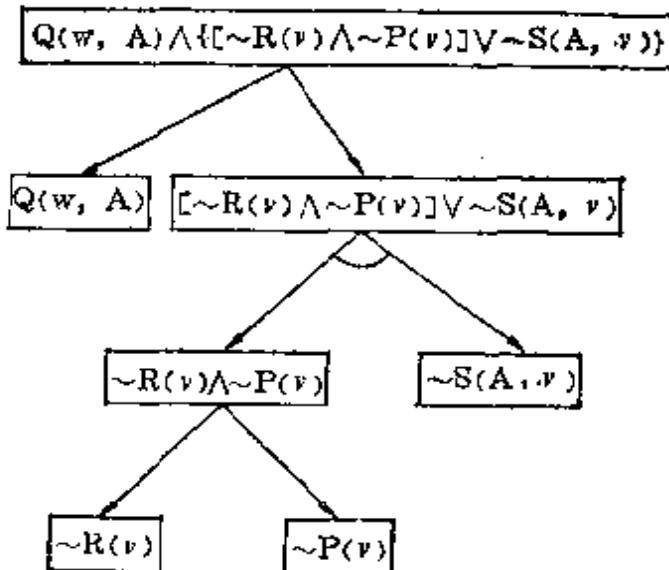


图 4.21 一事实表达式的一棵与或树表示

这样与或图的根节点就是整个事实表达式，端节点都是表达式中的每一个文字，图 4.21 就是上式的与或图表示。

有了与或图的表示，就可以求出其解图（结束在文字节点上的子图）集，我们可以发现解图集与子句集有一一对应关系，即可从一个解图读得一个子句，这样可得对应的三个子句为：

$$Q(w, A), \neg R(v) \vee \neg S(A, v), \neg P(v) \vee \neg S(A, v)$$

这个性质很有用，因此也可以把这种与或图看成是子句集的一种表示。

由于与或图表示中，合取元没有进一步展开，因此变量改名受到一定的限制，这一点是不如子句集表示灵活，所以一般性差一些。

应当指出，这里的与或图是作为综合数据库的一种表示，其中变量受全称量词的约束。而在可分解产生式系统中，所描述的与或图是搜索过程的一种表示，两者有不同的目的和含义，因此在应用时应加以区别。

2. 应用规则对与或图作变换

正向产生式系统是以正向方式使用规则 (F 规则) 对表示综合数据库的与或图结构进行变换，这些规则是以蕴涵形给出，在讨论应用它们进行推理之前，对规则形式的限制作一点说明。

为了讨论问题简单起见，设规则的左侧具有单文字的形式，即 $L \rightarrow W$ ，其中 L 是单文字， W 是任一化成与或形的公式。这个蕴涵式中的所有变量都假定有全称量词的约束，并且变量已经换名，使之与事实公式或其他规则公式中的变量区别开来。为了得到这种可应用的形式，对原始蕴涵式必须作必要的化简。设原始公式为

$$(\forall x)((\exists y)(\forall z)P(x, y, z)) \rightarrow (\forall u)Q(x, u))$$

则化简步骤如下：

(1) 暂时消去蕴涵符号

$$(\forall x)(\sim((\exists y)(\forall z)P(x, y, z)) \vee (\forall u)Q(x, u))$$

(2) 处理否定符号使其作用辖域仅限于单个文字

$$(\forall x)((\forall y)(\exists z)(\sim P(x, y, z)) \vee (\forall u)Q(x, u))$$

(3) Skolem 化

$$(\forall x)((\forall y)(\sim P(x, y, f(x, y))) \vee (\forall u)Q(x, u))$$

(4) 化成前束式并忽略去全部全称量词

$$\sim P(x, y, f(x, y)) \vee Q(x, u)$$

(5) 恢复蕴涵式表示

$$P(x, y, f(x, y)) \rightarrow Q(x, u)$$

对规则作单文字前项的限制将大大简化了应用时的匹配过程，对非单文字前项的情况，若形式为 $(L_1 \vee L_2) \rightarrow W$ ，则可化成与其等价的两个规则 $L_1 \rightarrow W$ 与 $L_2 \rightarrow W$ 进行处理。单文字的限制对实际问题的应用范围有所限制，但对演绎方法本身并不产生影响。下面分两种情况来讨论演绎的过程。

(1) 命题逻辑的情况

由于所有公式都不含有变量，因此应用规则的匹配过程比较简单。设初始综合数据库的与或形表达式为

$$((P \vee Q) \wedge R) \vee (S \wedge (T \vee U))$$

规则为

$$S \rightarrow (X \wedge Y) \vee Z$$

把初始数据库用与或图表示，图中有一个端节点是文字 S ，它正好与规则前项的文字 S 完全匹配，由此可直接用这条规则对与或图进行变换，即把规则后项的与或形公式用与或图表示后添加到初始数据库上，并用一个匹配弧连接起来，规则匹配后演绎的结果如图 4.22 所示（把根节点画在底部）。图中匹配弧后面是规则部分，其前项可看成匹配弧的一个后裔节点，由匹配上的文字标记，它表示成规则后项与或图结构的根节点。

这个应用一条规则之后演绎得到的新与或图，可以认为既可

表示原始事实，又能表示推出的事实表达式。如果撇开匹配弧后面的部分，那么就可以从初始与或图的4个解图中，得到对应的4个子句组成的子句集：

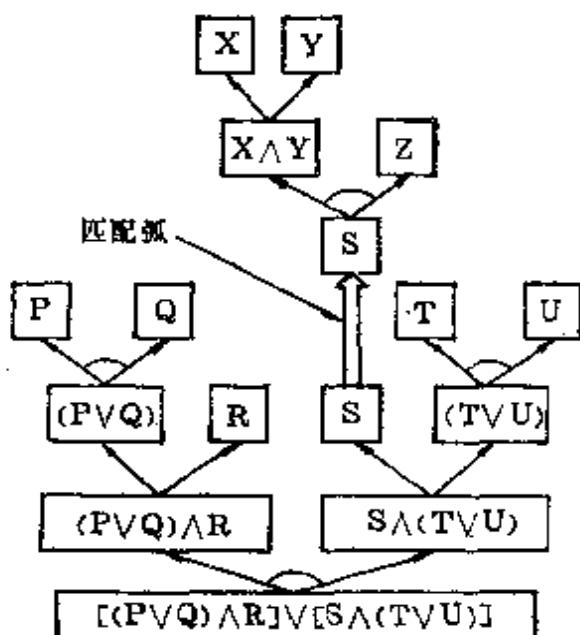


图 4.22 应用一条规则后的与或图的子句集：

$X \vee Z \vee P \vee Q$
 $X \vee Z \vee R$
 $Y \vee Z \vee P \vee Q$
 $Y \vee Z \vee R$
 $(T \vee U) \vee (P \vee Q)$
 $(T \vee U) \vee R$

可以看出前面4个新增加的子句就是初始事实和规则表示的子句一起组成的子句集进行归结可能得到的所有归结式，即演绎得到的子句集就是归结式的完备集。下面说明一下这个结论。

把规则化成子句表示后可得两个子句

$\neg S \vee X \vee Z$
 $\neg S \vee Y \vee Z$

于是基本子句集为

$$\{S \vee (P \vee Q), S \vee R, (T \vee U) \vee (P \vee Q), (T \vee U) \vee R, \\ \sim S \vee X \vee Z, \sim S \vee Y \vee Z\}$$

不难求得所有的4个归结式正好就是解图集读到的4个新子句。也就是说一条则规的应用结果，就得到归结式的完备集，即这条规则的应用演绎出所有的逻辑推论来。

一个基于规则的正向演绎系统，其演绎过程就是不断地调用匹配上的规则对与或图进行变换，直到生成的与或图含有目标表达式为止，也就是说要用目标公式作为系统的结束条件。正向系统的目标表达式要限制为文字析取形（子句形）的一类公式，当目标公式中有一个文字同与或图中某一个端节点所标记的文字匹配上时，和规则匹配时做法一样，通过匹配弧把目标文字添加到图上，这个匹配弧的后裔节点称为目标节点。这样当产生式系统演绎得到的与或图包含有目标节点的解图时，系统结束演绎，这时便推出了一个与目标有关的子句。下面的简例说明系统的推理过程。

例：事实表达式：

$A \vee B$

规则集： $A \rightarrow C \wedge D$

$B \rightarrow E \wedge G$

目标公式： $C \vee G$

应用完这两条规则之后，得到的与或图如图4.23所示，其中有一个解图是满足目标公式($C \vee G$)所建立的结束条件。可以看出这种产生式系统具有可交换的性质，因而可以使用不可撤回的控制策略求出解图。

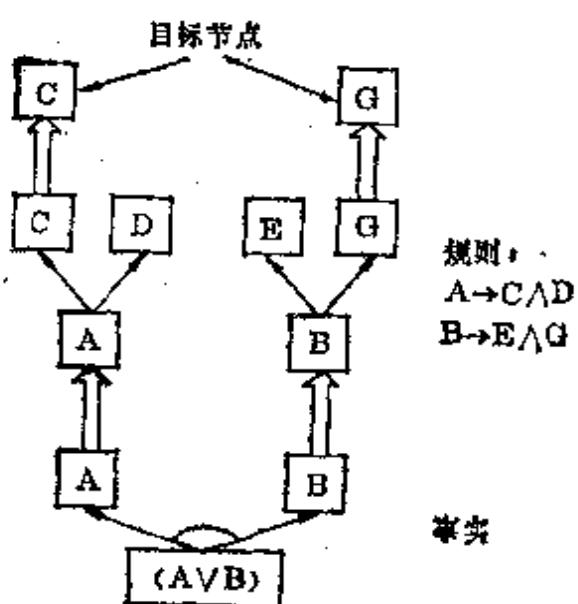


图 4.23 满足结束条件的与或图

(2) 谓词逻辑的情况

在谓词逻辑的情况下，对事实表达式和规则蕴涵式的处理过程在本节一开头已经进行了讨论，下面主要是讨论对含有存在量化变量或全称量化变量的目标公式应如何处理，以及规则应用时的匹配问题。

对具有量词量化变量的目标公式来说，化简时所使用的 Skolem 化过程是前面讨论过的事实和规则所用的 Skolem 化过程的对偶形式。即目标中属于存在量词辖域内的全称量化变量要用存在量化变量的 Skolem 函数来替代，经过 Skolem 化的公式只剩下存在量词，然后对析取元作变量改名，最后再把存在量词省略掉。

例如，设目标公式为

$$(\exists y)(\exists x)(P(x, y) \vee Q(x, y))$$

用 Skolem 函数消去全称量词后有

$$(\exists y)(P(f(y), y) \vee Q(f(y), y))$$

和命题逻辑的情况一样，目标公式也限制为文字的析取式，这时要进行变量改名，使每个析取元具有不同的变量符号，于是有

$$(\exists y)P(f(y), y) \vee (\exists y_1)Q(f(y_1), y_1)$$

最后忽略去存在量词后得

$$P(f(y), y) \vee Q(f(y_1), y_1)$$

以后目标公式中的变量都假定受存在量词的束约。

现在再来看一下应用一条规则 $L \rightarrow W$ 对与或图进行变换的过程。设与或图中有一个端节点的文字 L' 和 L 可合一， mgu 是 u ，则这条规则可应用，这时用匹配弧连接的后裔节点是 L ，它是规则后项 Wu 对应的与或图表示的根节点，在匹配弧上标记有 u ，表示用 u 替换后可与规则匹配。

例：事实与或形表示 $P(x, y) \vee (Q(x, A) \wedge R(B, y))$

规则蕴涵式 $P(A, B) \rightarrow (S(A) \vee X(B))$

图 4.24 是这个例子应用规则变换后得到的与或图，它有两个解图，对应的两个子句是

$$S(A) \vee X(B) \vee Q(A, A)$$

$$S(A) \vee X(B) \vee R(B, B)$$

它们正是事实和规则公式组成的子句集对文字 P 进行归结时得到的归结式。

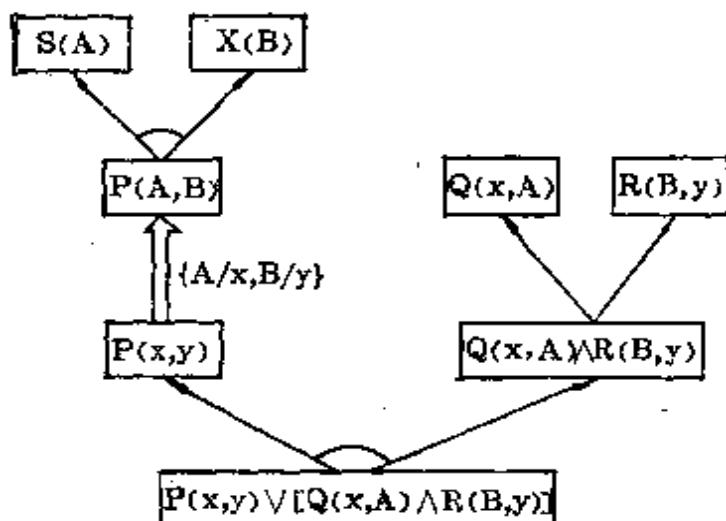


图 4.24 应用一条含有变量的规则后得到的与或图

当一个与或图应用了好几条规则之后，推出的与或图将含有多个的匹配弧，这时任一解图可能有多于一个的匹配弧（对应的置换是 u_1, u_2, \dots ），因此在列写解图的子句集时，只考虑具有致的匹配弧置换的那些解图（一致解图）。一个一致解图表示的子句是对得到的文字析取式应用一个合一复合的置换之后所得到的子句。

置换的一致集和置换的合一复合这两个概念定义如下。设有一个置换集 $\{u_1, u_2, \dots, u_n\}$ ，其中 $u_i = \{t_{i1}/v_{i1}, \dots, t_{im(i)}/v_{im(i)}\}$ 是置换对集合， t 是项， v 是变量。

根据这个置换集，再定义两个表达式：

$U_1 = (v_{11}, \dots, v_{1m(t_1)}, \dots, v_{n1}, \dots, v_{nm(t_n)})$ 由 u_1 的变量 v_i 构成

$U_2 = (t_{11}, \dots, t_{1m(t_1)}, \dots, t_{n1}, \dots, t_{nm(t_n)})$ 由 u_1 的项 t_i 构成

置换 (u_1, \dots, u_n) 称为一致的，当且仅当 U_1 和 U_2 是可合一的。

(u_1, \dots, u_n) 的合一复合 (unifying composition) u 是 U_1 和 U_2 的最一般的合一者。下面给出几个合一复合结果的实例。

u_1	u_2	U_1 和 U_2	合一复合 u
$\{A/x\}$	$\{B/x\}$	$U_1 = (x, x)$ $U_2 = (A, B)$	不一致
$\{x/y\}$	$\{y/z\}$	$U_1 = (y, z)$ $U_2 = (x, y)$	$\{x/y, x/z\}$
$\{f(z)/x\}$	$\{f(A)/x\}$	$U_1 = (x, x)$ $U_2 = (f(z), f(A))$	$\{f(A)/x, A/z\}$
$\{x/y, x/z\}$	$\{A/z\}$	$U_1 = (y, z, z)$ $U_2 = (x, x, A)$	$\{A/x, A/y, A/z\}$

可以证明对一个置换集求合一复合的运算是可结合和可交换的 (求置换的合成是不可交换的)，因此一个解图对应的合一复合不依赖于构造这个解图时所产生的匹配弧的次序。再强调一下，我们要求一个解图具有一个一致匹配弧的置换集，这样该解图所对应的子句才是从初始事实表达式和规则公式集推出的子句。

有些时候演绎过程会多次调用同一条规则，这时要注意每次应用都要使用改名的变量，以免匹配过程产生一些不必要的约束。此外也可多次使用同一目标文字来建立多个目标节点，这也需要采用改名的变量。

最后再举一个简单的例子来说明正向产生式系统的演绎过程。

设事实和规则描述如下：

Fido barks and bites, or Fido is not a dog.

$F_1: \sim\text{DOG(FIDO)} \vee (\text{BARKS(FIDO)} \wedge \text{BITES(FIDO)})$

All terriers are dogs.

$R_1: \sim\text{DOG}(x) \rightarrow \sim\text{TERRIER}(x)$

Anyone who barks is noisy.

$R_2: \text{BARKS}(y) \rightarrow \text{NOISY}(y)$

要证明的目标是

There exists someone who is not a terriers or who is noisy.

目标公式: $\sim\text{TERRIER}(z) \vee \text{NOISY}(z)$

公式中 x, y 是全称量词量化的变量, 而 z 是存在量词量化的变量。图 4.25 给出演绎得到的与或图, 图中结束在目标节点的一

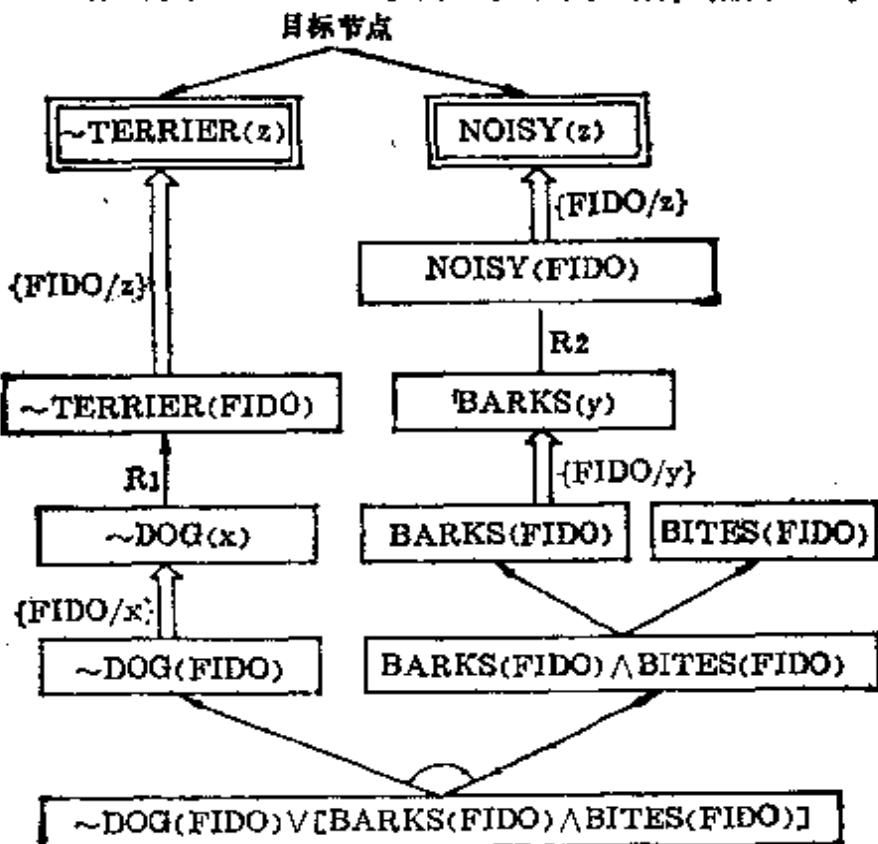


图 4.25 猎犬问题的与或图

一个一致解图, 有置换集 $\{\{FIDO/x\}, \{FIDO/y\}, \{FIDO/z\}\}$,

它的合一复合是 $u = \{FIDO/x, FIDO/y, FIDO/z\}$ 。根据这个一致解图，目标公式 $\neg TERRIER(z) \vee NOISY(z)$ 是事实和规则的逻辑推论，因而得到了证明。如果用这个合一复合 u 应用于这个目标公式，可得

$$\neg TERRIER(FIDO) \vee NOISY(FIDO)$$

它是已证目标公式的例，可作为一个回答语句。关于求解一致解图的搜索策略等问题，将留待下一节讨论。

4.8 基于规则的逆向演绎系统

逆向演绎系统中，是从目标表达式出发，反方向使用规则（B 规则）对目标表达式的与或图进行变换，最后得到含有事实节点的一致解图。整个推理过程的思路类同于正向系统，但具体的处理过程有其特点，本节主要介绍逆向系统的限制条件及处理方法。

1. 目标表达式的与或形表示

逆向系统能处理任意形式的目标表达式，在把任意目标公式化简为与或形的表示时，要使用与正向系统中对事实表达式处理方法的对偶形式。即要用存在量词量化变量的 Skolem 函数来替代全称量词的变量，消去全称量词，省略去存在量词，并进行变量换名使主析取元之间具有不同的变量名。对化简得到的与或形表达式，用与或图表示时，规定子表达式间的析取关系用 1-连接符来连接，表示成或的关系，而子表达式间的合取关系则用 k-连接符来连接，表示成与的关系。下面给出一个例子的化简结果及其与或图的表示。

例：目标公式： $(\exists y)(\forall x)(P(x) \rightarrow (Q(x, y) \wedge \neg(R(x) \wedge S(y))))$

Skolem 化后: $\sim P(f(y)) \vee (Q(f(y), y) \wedge (\sim R(f(y)) \vee \sim S(y)))$

变量标准化: $\sim P(f(z)) \vee (Q(f(y), y) \wedge (\sim R(f(y)) \vee \sim S(y)))$

这个目标的与或图表示如图 4.26, 其子句集可以从结束在端节点的解图集中读得:

$\sim P(f(z))$

$Q(f(y), y) \wedge \sim R(f(y))$

$Q(f(y), y) \wedge \sim S(y)$

这里规定子句是文字的合取式, 而这些子句的析取式是目标公式的子句形式(析取范式)。目标公式的与或图中, 根节点的任一后裔都称为一个子目标节点, 其中的表达式称为子目标。

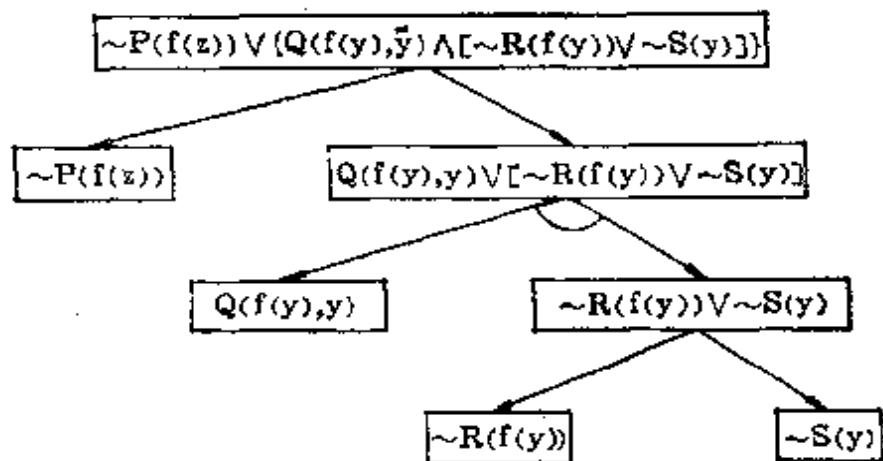


图 4.26 目标公式的与或图表示

2. 规则的应用

逆向系统中的 B 规则限制为 $W \rightarrow L$ 的形式, 其中前项 W 是任意形式的与或形公式, 后项 L 是单文字, 蕴涵式的任何变量都受全称量词约束。当 B 规则为 $W \rightarrow L_1 \wedge L_2$ 时, 则可化简为两条

规则 $W \rightarrow L_1$ 和 $W \rightarrow L_2$ 来处理。

当与或图中有某个端节点的文字 L' 和 L 可合一且 mgu 为 u 时，则 B 规则可应用，通过匹配弧连接的后裔节点 L ，就是规则前项 Wu 对应的与或图表示的根节点。规则应用后得到解图集所对应的子句就是对偶系统归结时得到的归结式集。即将规则 $W \rightarrow L$ 的否定式 ($W \wedge \neg L$) 得到的子句和目标公式的子句一起，对文字 L 进行归结得到的归结式。

逆向系统演绎过程的结束条件是生成的与或图含有事实表达式，而事实表达式限制为文字的合取形式。当事实表达式有一个文字同与或图中某一个端节点所标记的文字（子目标）匹配上时，就可以通过匹配弧把事实文字加到图上。这样当最后得到的与或图包含一个结束在事实节点上的一致解图时，系统便结束演绎，一个一致解图是解图中匹配弧置换集具有合一复合置换的那个解图。和正向系统类似，B 规则可以多次调用，事实文字也可以重复多次匹配，但每次匹配都要进行变量改名。下面通过一个简例说明逆向系统的演绎过程。

设事实有

- $F_1: \text{DOG(FIDO)}$
- $F_2: \neg \text{BARKS(FIDO)}$
- $F_3: \text{WAGS-TAIL(FIDO)}$
- $F_4: \text{MEOWS(MYRTLE)}$

规则集：

- $R_1: (\text{WAGS-TAIL}(x_1) \wedge \text{DOG}(x_1)) \rightarrow \text{FRIENDLY}(x_1)$
- $R_2: (\text{FRIENDLY}(x_2) \wedge \neg \text{BARKS}(x_2)) \rightarrow \neg \text{AFRAID}(y_2, x_2)$
- $R_3: \text{DOG}(x_3) \rightarrow \text{ANIMAL}(x_3)$
- $R_4: \text{CAT}(x_4) \rightarrow \text{ANIMAL}(x_4)$
- $R_5: \text{MEOWS}(x_5) \rightarrow \text{CAT}(x_7)$

询问：

If there are a cat and a dog such that the cat is
unafraind of the dog.

目标公式：

$$(\exists x)(\exists y)(\text{CAT}(x) \wedge \text{DOG}(y) \wedge \neg \text{AFRAID}(x, y))$$

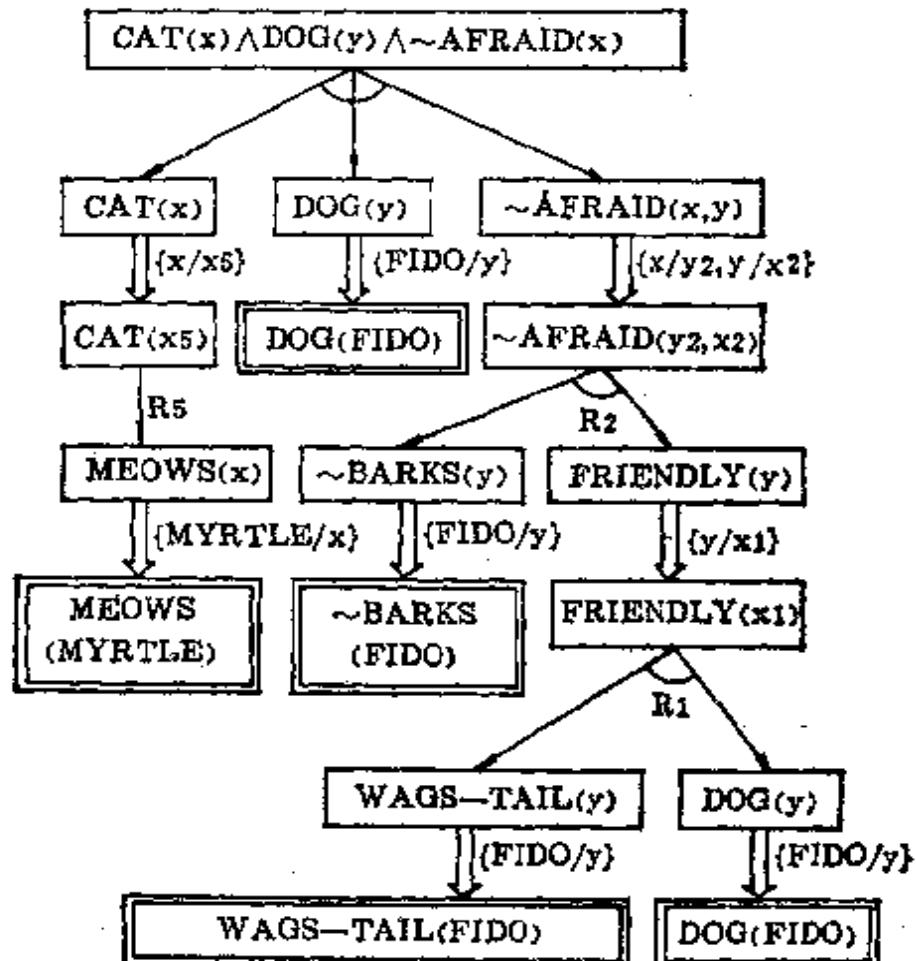


图 4.27 逆向系统的一致解图

图 4.27 给出这个问题逆向求解的一个一致解图，图中事实节点由双框线标记，规则的应用由规则编号标记。解图中的所有匹配弧的置换集是

$$\begin{aligned} &\{\{x/x_5\}, \{MYRTLE/x\}, \{FIDO/y\}, \{x/y_2, y/x_2\}, \\ &\{FIDO/y\}, \{y/x\}, \{FIDO/y\}, \{FIDO/y\} \} \end{aligned}$$

由此求得的合一复合是

{MYRTLE/x₅, MYRTLE/x, FIDO/y, MYRTLE/y₂,
FIDO/x₂, FIDO/x₁}

解图是一个一致解图，目标公式得到证明。把这个合一复合置换应用到目标公式得到的例，其回答语句为

(CAT(MYRTLE) \wedge DOG(FIDO) \wedge ~
AFRAID(MYRTLE, FIDO))

3. 演绎系统的搜索策略

演绎系统的搜索目标是找一个一致解图，其搜索策略的基本思想是首先找一个任意解图，再检验其一致性，看是否是一致解图。如果这个候选解图不一致，则继续搜索直到找到一个一致解图为止。下面介绍两种逆向系统的搜索方法，但其思想也可用于正向系统的搜索过程。

(1) 修剪不一致的局部候选解图

在扩展局部的候选解图前，就进行一致性的检验，若能及时发现不一致的局部候选者并立即修剪掉，则将减少无用的搜索，致使效率提高。下面通过两个例子来说明这一问题。

例 1：设要证明的目标： $P(x) \wedge Q(x)$

规则： $R_1: R(y) \rightarrow P(y)$

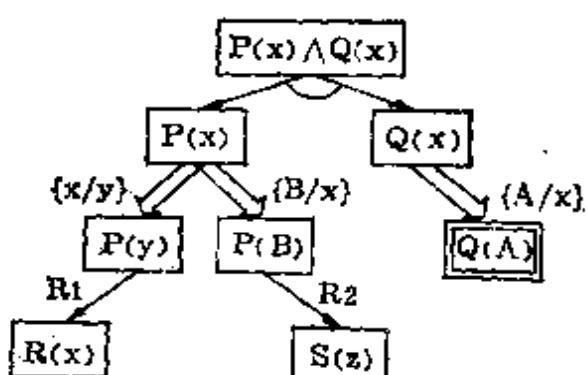


图 4.28 例1的与或图

$R_2: S(z) \rightarrow P(B)$

事实： $R(A) \wedge Q(A)$

图 4.28 是已生成的一个与或图，其中包含两个局部的候选解图，这两个候选者所对应的匹配置换集分别为 $\{\{x/y\}, \{A/x\}\}$ 和 $\{\{B/x\}, \{A/x\}\}$ ，显然后者

是不一致置换。如果 $Q(A)$ 是子目标 $Q(x)$ 的唯一匹配，则 R_2 就不可能是任意一个解的一部分，因此必须中止 R_2 以下部分的搜索，即不必要再生成 $S(z)$ 的一些子目标。

例 2：

目标： $P(x, x)$

规则： $R_1: (Q(u) \wedge R(v)) \rightarrow P(u, v)$

$R_2: W(y) \rightarrow R(y)$

$R_3: S(w) \rightarrow R(w)$

$R_4: U(z) \rightarrow S(C)$

$R_5: V(A) \rightarrow Q(A)$

设演绎生成的部分与或图如图 4.29 所示，其中一个应用了 R_4 和 R_5 的局部解图是不一致的，有两个置换 $\{A/x\}$ 和 $\{C/x\}$ 是不一致的。如果 R_5 对子目标 $Q(x)$ 是唯一的匹配，则应修剪掉子目标 $U(z)$ ，而子目标 $S(x)$ 仍保留并允许作置换 $\{A/x\}$ 。

(2) 建立规则连接图结构

这个方法是预先计算规则间所有可能的匹配，组成规则连接图，并存储产生的各个置换，在求解具体问题再调用这些结果。只要规则集不变，这些结果对所有具体问题都有效，当然只有对不太大的规则集来说，这个方法才是实际的。

图 4.30 是用前面猫和狗例题的规则集构造的规则连接图。构造的办法是先把每条规则的与或图表示画好，然后把一条规则前项的某一个文字通过匹配弧与另一条规则的后项文字连接起

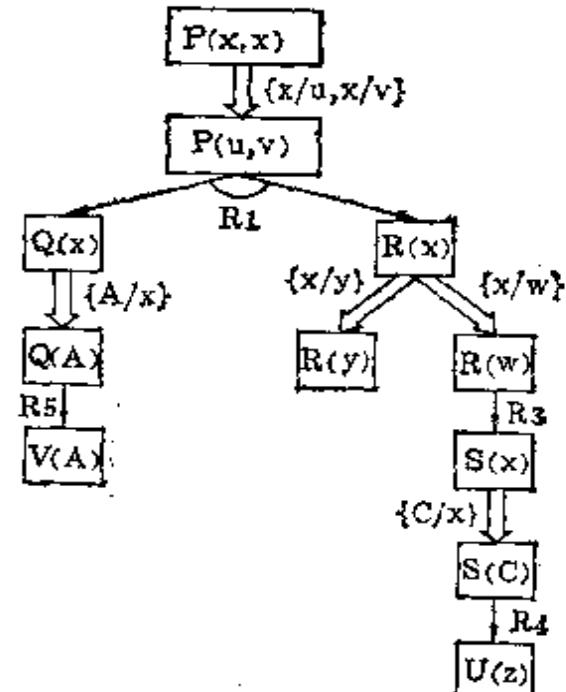


图 4.29 例2的与或图

来，并在匹配弧上标上 mgu。

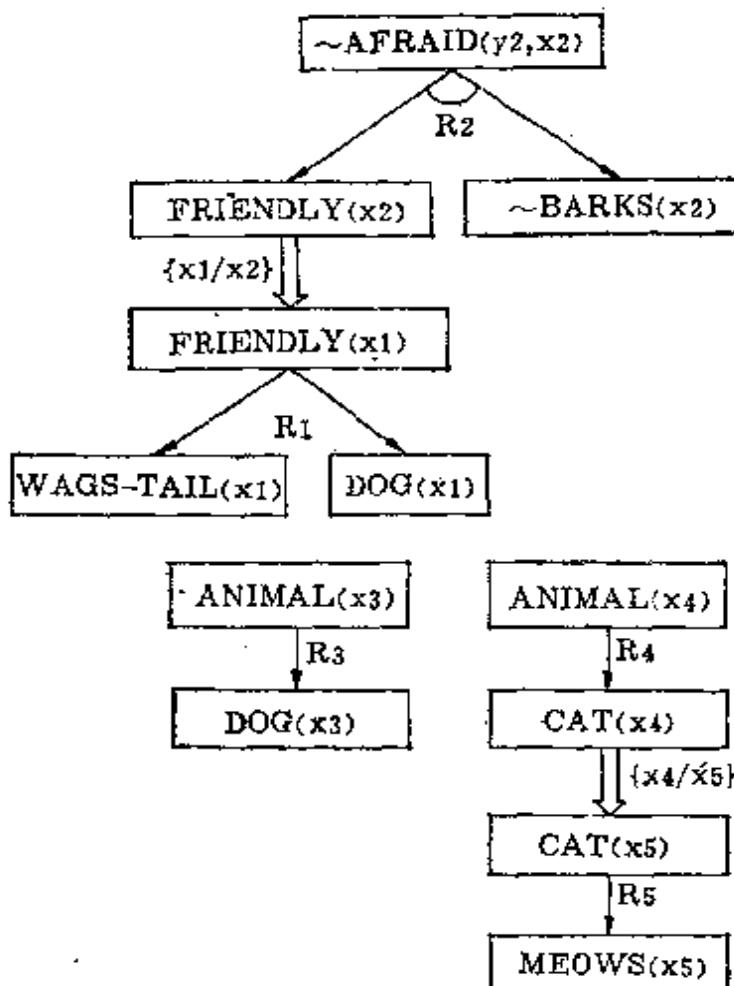


图 4.30 一个规则连接图

当要求解一个实际问题时，把目标与或图中的各个文字节点连接到某一条规则后项的文字上（对应的文字能匹配），把各个事实节点连接到某一条规则前项的文字上（对应的文字能匹配），这样就实现了目标节点和事实节点同规则连接图相结合的扩大连接图。接着就寻找其中的一个候选解图，一旦找到候选者，就计算其置换集的合一复合。若存在，就找到一个一致解图，从而得到问题的解，否则就必须再查找该连接图另外的候选解图。

这种方法实际上根据预先计算的连接图结构，不断地产生愈

来愈大的一系列与或图。一个很复杂的问题是可能要在候选解图中多次使用同一条规则的连接图，每次使用时变量都要改名，因而这些改名变量也必须出现在候选解图的置换之中。下面举一个特殊的例子来说明多次使用规则连接图的复杂性。

设有规则 $P(x) \rightarrow P(f(x))$ 和事实 $P(A)$ ，要证明目标公式 $P(f(f(A)))$ 。这个问题的规则连接图如图 4.31 所示，图中有一个规则前项和后项的匹配弧，但未加置换标记，以便引起注意该规则的某一个新例能使其后项和原始的前项匹配等等。当目标节点和事实节点和规则连接图连结时，可得连接图如图 4.32 所示。根据这个连接图，若两次使用这条规则（即沿匹配弧循环一次），则可得一个如图 4.33 所示的候选解图。由于两次使用同一条规则，因而规则中变量和连接匹配弧的置换变量都必须改名。根据图 4.33 中匹配置换集可求得合一复合置换为 $\{f(A)/x, A/y\}$ ，所以这个解图是一致解图。

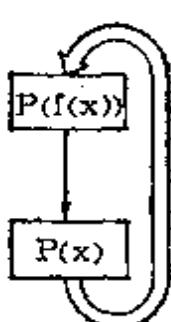


图 4.31 例子的
规则连接图

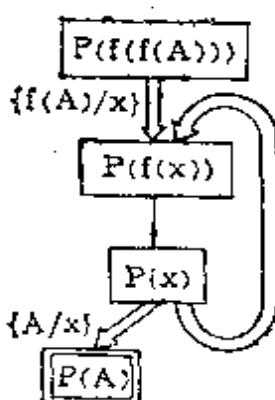


图 4.32 一个连接图

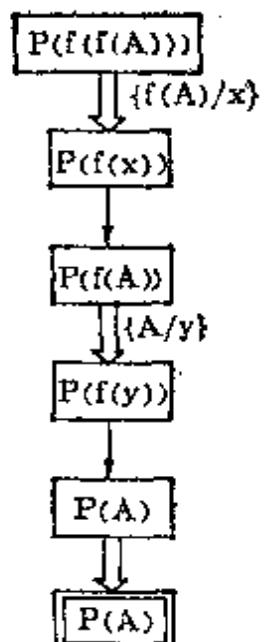


图 4.33 一个候选解图

4.9 基于规则的演绎系统的几个问题

1. 正向系统和逆向系统特点对比

	正向系统	逆向系统
使用条件	(1) 事实表达式是任意形式 (2) 规则形式为 $L \rightarrow W$ 或 $L_1 \vee L_2 \rightarrow W$ (L 为单文字, W 为任意形式) (3) 目标公式为文字析取形	(1) 事实表达式是文字合取形式 (2) 规则形式为 $W \rightarrow L$ 或 $W \rightarrow L_1 \wedge L_2$ (L 为单文字, W 为任意形式) (3) 目标公式是任意形式
化简过程	(1) 用Skolem函数消去事实表达式中的存在量词, 化简的公式受全称量词的约束 (2) 对规则的处理同(1) (3) 用Skolem函数(对偶形)消去目标公式中的全称量词, 化简的公式受存在量词约束	(1) 用Skolem函数(对偶形)消去目标公式中的全称量词, 化简的公式受存在量词的约束 (2) 对规则的处理同(3) (3) 用Skolem函数消去事实表达式中的存在量词, 化简的公式受全称量词的约束
初始综合数据库	事实表达式的与或树 (\vee 对应为与关系, \wedge 对应为或关系)	目标公式的与或树 (\vee 对应为或关系, \wedge 对应为与关系)
推理过程	从事实出发, 正向应用规则(变量改名, 前项与事实文字匹配, 后项代替前项), 直至得到目标节点为结束条件的一致解图为止	从目标出发, 逆向应用规则(变量改名, 后项与子目标文字匹配, 前项代替后项), 直至得到事实节点为结束条件的一致解图为止
子句形式	文字的析取式	文字的合取式
子集形式	子句的合取式(合取范式)	子句的析取式(析取范式)

虽然正向系统中的规则和目标公式以及逆向系统的事实公式和规则都加有限制, 但它们仍然可以有许多情况适用。我们还可以把正向和逆向的推理结合在一起, 建立基于规则的双向演绎系

统，这样可减少使用的限制，但在处理结束条件以及 F 规则和 B 规则的选取策略方面比较复杂，这方面的问题可参阅有关参考文献。

2. 关于控制知识

一般人工智能系统所用到的知识分为三种类型：陈述知识、过程知识和控制知识。对于基于规则的演绎系统来说，比较容易表达陈述知识和过程知识，因为用谓词形式和蕴涵形规则来表示领域的专业知识比较方便和自然。另一方面，为了提高系统的求解效率，仍然有必要提供足够的控制知识，因为有效的搜索策略要求有控制知识来引导搜索。演绎系统中有几种控制信息是比较有效的。例如如何在最好的推理方向上应用一条规则，因为有的问题可能在某个方向应用规则会导致低效率，一般推理方向应尽可能选在分枝选择数目较小的方向上。还有在对某一条 B 规则的前项或 F 规则的后项，领域专家可能会规定其子表达式处理的优先顺序。再者，有时候对规则规定一些附加的应用条件来限制规则的使用也会提高效果，这都是一些重要的控制知识。至于使用这类控制知识问题，有一些可行的方法：如可把有关的控制知识直接嵌入到规则之中，这需要编程技巧；也可以把控制知识嵌入到搜索策略所使用的评价函数之中；再复杂一点就是建立一个产生式系统来求解搜索策略中的问题。总之关于控制知识问题，在实现具体系统时都需要做进一步的研究才能解决。

4.10 小 结

1. 一阶谓词演算在人工智能系统中可作为知识表示的工具，其演绎推理方法可用来建立自动定理证明系统、问答系统和基于规则的演绎系统等。在定理证明系统中，反演法简单有效，系统

不是直接证明 $F_1 \wedge F_2 \wedge \cdots \wedge F_n \rightarrow W$ 为永真，而是去证明 $F = F_1 \wedge F_2 \wedge \cdots \wedge F_n \wedge \sim W$ 为永假，这等价于证明公式 F 对应的 Skolem 化标准形子句集 $S = S_0 \cup \{\sim W\}$ 为不可满足的。

2. 归结反演系统是应用归结（消解）原理建立的产生式系统，它把子句集 S 作为初始数据库，不断应用归结推理规则生成出归结式来，当出现空子句（目标条件）的归结式时过程结束，这说明了 S 是不可满足的，从而使定理得到证明。

实现归结过程的搜索策略有支持集策略、单元子句优先策略、线性输入形策略和祖先过滤策略等，主要是解决提高搜索空子句的效率问题。

3. 可以利用归结反演过程来提取问答，方法是先将询问表示成要证明的目标公式，再根据反演法求归结反演树，最后按反演树的过程构造修改证明树，其根部的子句就是要提取的回答。

4. 基于规则的演绎系统是一个直接证明系统，主要强调直接使用规则进行演绎，以便使求解过程易于为人们所理解和接受。

正向演绎系统是从事实表达式出发，通过正向匹配应用规则进行推理，直到推出目标表达式。演绎过程是把事实表达式化为与或形表示，并按规定的与或关系表示成与或树形式来进行。

逆向演绎系统是从目标表达式出发，通过逆向匹配应用规则进行推理，直到推出事实表达式。演绎过程也是把目标表达式化为与或形表示，其与或关系是正向系统的对偶形式，其他过程类同。

5. 一般来说，归结法也是一种问题的求解方法。可以把需要求解的问题用定理证明的形式来表示，然后用归结法证明，并通过修改证明树提取回答，从而可得到问题的解。

习 题

4.1 化下列公式成子句形式：

- (1) $(\forall x)[P(x) \rightarrow P(x)]$
- (2) $\{\sim\{(\forall x)P(x)\}\} \rightarrow (\exists x)[\sim P(x)]$
- (3) $\sim(\forall x)\{P(x) \rightarrow \{(\forall y)[P(y) \rightarrow P(f(x, y))]\}$
 $\quad \wedge \sim(\forall y)[Q(x, y) \rightarrow P(y)]\}$
- (4) $(\forall x)(\exists y)\{[P(x, y) \rightarrow Q(y, x)] \wedge [Q(y, x) \rightarrow S(x, y)]\} \rightarrow (\exists x)(\forall y)[P(x, y) \rightarrow S(x, y)]$

4.2 以一个例子证明置换的合成是不可交换的。

4.3 找出集 $\{P(x, z, y), P(w, u, w), P(A, u, u)\}$ 的mgu。

4.4 说明下列文字集不能合一的理由：

- (1) $\{P(f(x, x), A), P(f(y, f(y, A)), A)\}$
- (2) $\{\sim P(A), P(x)\}$
- (3) $\{P(f(A), x), P(x, A)\}$

4.5 已知两个子句为

$Loves(father(a), a)$
 $\sim Loves(y, x) \vee Loves(x, y)$

试用合一算法求第一个子句和第二个子句的第一个文字合一的结果。

4.6 用归结反演法证明下列公式的永真性：

- (1) $(\exists x)\{[P(x) \rightarrow P(A)] \wedge [P(x) \rightarrow P(B)]\}$
- (2) $(\forall z)[Q(z) \rightarrow P(z)] \rightarrow \{(\exists x)[Q(x) \rightarrow P(A)] \wedge [Q(x) \rightarrow P(B)]\}$
- (3) $(\exists x)(\exists y)\{[P(f(x)) \wedge Q(f(B))] \rightarrow [P(f(A)) \wedge P(y) \wedge Q(y)]\}$
- (4) $(\exists x)(\forall y)P(x, y) \rightarrow (\forall y)(\exists x)P(x, y)$
- (5) $(\forall x)\{P(x) \wedge [Q(A) \vee Q(B)]\} \rightarrow (\exists x)[P(x) \wedge Q(x)]$

4.7 以归结反演法证明公式 $(\exists x)P(x)$ 是 $[P(A_1) \vee P(A_2)]$ 的逻辑推论，然而， $(\exists x)P(x)$ 的 Skolem 形即 $P(A)$ 并

非 $[P(A_1) \vee P(A_2)]$ 的逻辑推论, 请加以说明。

4.8 给定下述语句:

John likes all kinds of food.

Apples are food.

Anything anyone eats and isn't killed by is food.

Bill eats peanuts and is still alive.

Sue eats everything Bill eats.

(1) 用归结法证明“John likes peanuts.”

(2) 用归结法提取回答“What food does Sue eat?”

4.9 已知事实公式为

$$(\forall x)(\forall y)(\forall z)(Gt(x, y) \wedge Gt(y, z) \rightarrow Gt(x, z))$$

$$(\forall a)(\forall b)(Succ(a, b) \rightarrow Gt(a, b))$$

$$(\forall x)(\sim Gt(x, x))$$

求证 $Gt(5, 2)$

试判断下面的归结过程是否正确? 若有错误应如何改进。

$$\sim Gt(5, 2) \quad \sim Gt(x, y) \vee \sim Gt(y, z) \vee Gt(x, z)$$

$$\swarrow \{5/x, 2/z\}$$

$$\sim Gt(5, y) \vee \sim Gt(y, 2) \quad \sim Succ(a, b) \vee Gt(a, b)$$

$$\swarrow \{y/a, 2/b\}$$

$$\sim Gt(5, y) \vee \sim Succ(y, 2) \quad \sim Gt(x, y) \vee \sim Gt(y, z) \vee Gt(x, z)$$

$$\swarrow \{5/x, y/z\}$$

$$\sim Gt(5, y) \vee \sim Gt(y, y) \vee \sim Succ(y, 2)$$

|

T

4.10 设公理集为

$$(\forall u)LAST(\text{cons}(u, NIL), u) \quad (\text{cons} \text{是表构造函数})$$

$$(\forall x)(\forall y)(\forall z)(LAST(y, z) \rightarrow LAST(\text{cons}(x, y), z))$$

(LAST(x, y) 代表 y 是表 x 的最末元素)

(1) 用归结反演法证明如下定理:

($\exists v$) LAST(cons(2, cons(1, NIL)), v)

(2) 用回答提取过程求表 (2,1) 的最末元素 v 。

(3) 简要描述如何使用这个方法求长表的最末元素。

4.11 对一个基于规则的几何定理证明系统, 把下列语句表示成产生式规则:

(1) 两个全等的三角形的对应角相等。

(2) 两个全等的三角形的对应边相等。

(3) 如果两个三角形对应边是相等的, 则这两个三角形全等。

(4) 一个等腰三角形的底角是相等的。

4.12 我们来考虑下列一段知识: Tony, Mike 和 John 属于 Alpine 俱乐部, Alpine 俱乐部的每个成员不是滑雪运动员就是一个登山运动员, 登山运动员不喜欢雨而且任一不喜欢雪的人不是滑雪运动员, Mike 讨厌 Tony 所喜欢的一切东西, 而喜欢 Tony 所讨厌的一切东西, Tony 喜欢雨和雪。

以谓词演算语句的集合表示这段知识, 这些语句要适合一个逆向的基于规则的演绎系统。试说明这样一个系统怎样才能回答问题“有没有 Alpine 俱乐部的一个成员, 他是一个登山运动员但不是一个滑雪运动员呢?”

4.13 一个积木世界的状态由下列公式集描述:

ONTABLE(A) CLEAR(E)

ONTABLE(C) CLEAR(D)

ON(D, C) HEAVY(D)

ON(B, A) WOODEN(B)

HEAVY(B) ON(E, B)

绘出这些公式打算要描述这个状态的草图。

下列语句提供了有关这个积木世界的一般知识：

每个大的蓝色积木块是在一个绿色积木块上。

每个重的木制积木块是大的。

所有顶上没有东西的积木块都是蓝色的。

所有木制积木块是蓝色的。

以具有单文字后项的蕴涵式的集合表示这些语句。绘出能求解“哪个积木块是在绿积木块上”这个问题的一致与或解图（用 B 规则）。

第五章 人工智能系统规划方法

上几章我们主要讨论了人工智能问题求解的基本方法——图搜索法、分解法（归约法）和归结证明法，所涉及的一些应用例子也多为一些智力游戏的典型问题。实际上有时候同一个问题，这几种方法都可使用，只是根据问题的特点，可能有的方法更简捷适用而已。但必须指出，人工智能中有许多复杂的实际问题，必须综合利用这些方法和策略以及多种知识表示技术，才能较好地加以解决。本章将通过积木世界机器人问题求解为示范，来讨论复杂问题求解的规划方法问题。

5.1 规划 (Planning)

在图搜索法中，我们是把问题求解过程描述成状态空间的搜索，搜索算法中，每一个所考察的节点代表一个完整的问题状态描述，而每一条规则则阐明改变总体状态描述的一种方法。显然对八数码游戏等这类问题，对完整的状态描述进行处理是比较容易而且较为合理。然而对较复杂的问题领域，求解时应从两个方面进行考虑。一个是有归约法，把原本问题分解成若干小的子问题分别进行求解，最后再把每一个子问题的解组合在一起构成原本问题的完整解，这样可简化求解过程。但要注意到，有些问题可以分解为相互独立的子问题，而有的往往不行，然而有许多问题则可近似看成是可分解的，即把问题分解为相互影响较少的子问题，在求解过程再对相互作用的问题采用一些专门的办法进行处理。另一点是当从一个问题状态转换到另一个状态时，不必重

新计算全部的新状态，而只考虑那些可能发生变化的部分，这就是所谓的画面问题(frame problem)。形象的比喻就是一个状态描述和另一个状态描述之间的变化好象动画片中一个镜头到另一个镜头画面的变化那样，很多的背景布局不变，只是角色动作涉及的局部场景发生变化，因此只需考虑变化部分的状态描述及其相应的操作规则，这样也可简化处理过程，达到能求解复杂问题的目的。

人工智能的研究中，已经提出了若干处理这两种化简的方法，这些方法着重于解决如何将原本问题分解成合适的子问题以及在求解过程中如何记录并处理子问题间相互作用的问题。因此和以前讨论过的搜索和推理过程一样，规划过程也是一种问题求解方法，一个规划系统就是一个问题求解系统，若用它求解比较复杂的问题，即可求得一个动作序列（串行、并行或串并行），来完成一定的具体任务，最终达到某一个目标。另一方面，就规划这个词来说，其含意是指在求解实际问题时，在执行一系列动作之前，事先要制定出求解过程的一系列步骤，即先作出规划，然后再付之实施。而规划的作用不仅限于动作前给出动作执行的先后次序，而且还可用于监视动作序列的执行过程，一旦发现不应出现的意外情况，并影响到动作的继续执行时便能及时作出处理。在现实世界中，人们求解复杂实际问题的过程是事先尽可能地作周密的考虑，处理好各具体任务之间的相互干扰，制定出可行的规划，然后按步执行。在执行中往往还会出现问题或意外情况，因此要对尚未执行部分进行及时修正（甚至要重新规划），直至达到目标为止。人工智能中的规划方法实际上就是模拟人类求解复杂问题的这一过程。

5.2 机器人问题求解

机器人是一种能灵活地完成特定的操作和运动任务并可再编程序的多功能操作器，通常模拟人类动作和行为的机械电子装置统称为机器人。目前机器人有两类，一类是工业机器人，它是一种按照事先编好的程序，重复进行操作的装置；另一类是智能机器人，是利用感觉器官对环境进行识别和理解，然后作出决策和行动的高级操作装置。机器人问题求解是在机器人世界环境中，察看周围世界，作出行动规划，并监督规划的实际执行。机器人问题求解是一个很复杂的综合性问题，本章只是对积木世界中的简单问题，通过机器人动作序列的规划过程，来阐明规划的基本概念和方法。

1. 机器人问题描述

该机器人有一只机械手，处理的世界有一张桌子，可堆放若干相同的方积木块，机械手有4个操作积木的典型动作：从桌上拣起一块积木；将手上的积木放到桌上；搭起一摞积木（摞上一块）；拆掉一块（拿下一块）积木。如一个积木世界的布局如图5.1所示，要求实现的目标是：把B放到C上，A放到B上，则

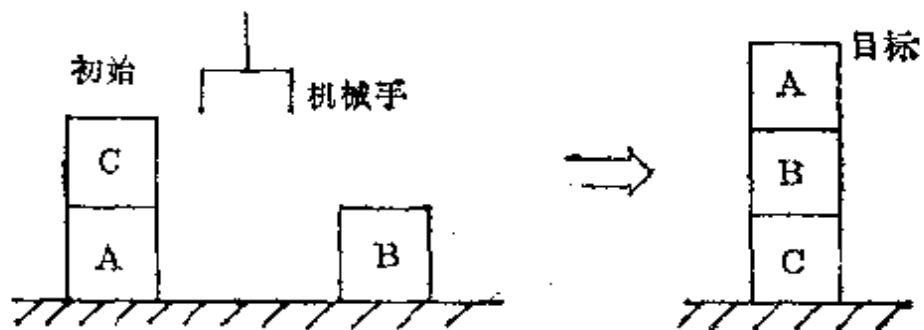


图 5.1 简单积木世界布局

问题很容易用谓词逻辑公式描述如下：

初始状态

CLEAR(B), ON(C, A), ONTABLE(A)

CLEAR(C), HANDEMPTY, ONTABLE(B)

目标状态

ON(B, C) \wedge ON(A, B)

一般情况下，可用任意有限个谓词公式的合取式来描述不同的世界状态。为简化以后的讨论，假设对目标（和子目标）表达式限制为文字的合取式，且变量均由存在量词量化；对初始状态或其他中间状态描述，则限制为基文字（无变量）的合取式。

机器人的动作是要改变世界的状态或布局，有一种描述是STRIPS系统（一个机器人问题求解系统）的形式化方法。用STRIPS型的F规则来模拟机器人的动作简单有效，其表示形式为

pickup(x)

P: ONTABLE(x) \wedge HANDEMPTY \wedge CLEAR(x)

D: ONTABLE(x), HANDEMPTY

A: HOLDING(x)

putdown(x)

P: HOLDING(x)

D: HOLDING(x)

A: ONTABLE(x) \wedge CLEAR(x) \wedge HANDEMPTY

stack(x, y)

P: HOLDING(x) \wedge CLEAR(y)

D: HOLDING(x), CLEAR(y)

A: HANDEMPTY \wedge ON(x, y) \wedge CLEAR(x)

unstack(x, y)

P: HANDEMPTY \wedge CLEAR(x) \wedge ON(x, y)

D: HANDEMPTY, ON(x, y)

A: HOLDING(x) \wedge CLEAR(y)

这种型式的 F 规则由三部分组成：

(1) 先决条件公式 P。这部分相当蕴涵规则的前项，并假定先决条件是文字的合取式，公式中的变量都假定有存在量词量化。为了使 F 规则可应用于某一状态描述，必须使其先决条件的某一置换的例能与当前状态相匹配，即若事实公式中有与先决条件中每一个文字都能合一的文字，且全部 mgu 是一致的（有合一复合），则先决条件与事实匹配，并称该合一复合为匹配置换，通过这个置换就得到 F 规则的例，这条规则就可以应用。此外，对于给定的一条 F 规则和一个状态描述，可以有多种匹配置换，每一个置换都可以得到一个 F 规则可应用的例。

(2) 删除文字表 D。删除表的文字可包含自由变量，当一条 F 规则应用到一个状态描述时，匹配置换应用到删除表的各文字上，这样得到的基例就从原来的状态描述中被删去，作为构造新状态的一个步骤。这里再假定删除表中的所有自由变量和先决条件中的变量相同，这个限制使删除表文字任一匹配的例都是一个基文字。

(3) 加添公式 A。这部分是由文字（可能含有自由变量）的合取式组成，相当于蕴涵规则的后项。当一条 F 规则应用到一个状态描述时，匹配置换应用到加添公式上，这样得到的匹配例可加到原来的状态描述中，作为构造新状态的第二个步骤。这里同样假定 A 中的自由变量与先决条件的变量相同，这样可使加添公式任一匹配的例都是基文字的合取。

这里还要指出，积木世界状态和机器人动作的描述也可用另一种谓词逻辑形式描述，例如 Green 系统中，对图 5.1 的给定状态 s_0 可表示为如下一组谓词公式（定理）：

$$\text{ON}(C, A, s_0) \wedge \text{ONTABLE}(A, s_0) \wedge \text{CLEAR}(C, s_0)$$
$$\wedge \text{ONTABLE}(B, s_0) \wedge \text{CLEAR}(B, s_0) \wedge \text{HANDEMPTY}(s_0)$$

对动作 $\text{unstack}(x, y)$ 可用如下公理描述：

$$\begin{aligned}
 & (\text{CLEAR}(x, s) \wedge \text{ON}(x, y, s)) \rightarrow (\text{HOLDING}(x, \\
 & \quad \text{do}(\text{unstack}(x, y), s)) \wedge \text{CLEAR}(y, \\
 & \quad \text{do}(\text{unstack}(x, y), s)))
 \end{aligned}$$

其中 do 函数确定了执行动作生成的新状态，公理表明在状态 s 中， $\text{CLEAR}(x)$ 和 $\text{ON}(x, y)$ 成立，那么 $\text{HOLDING}(x)$ 和 $\text{CLEAR}(y)$ 在新状态下成立，该新状态是在 s 状态中执行了 $\text{unstack}(x, y)$ 后获得的。

这样利用定理和公理便可证明 $\text{HOLDING}(C, s_1) \wedge \text{CLEAR}(A, s_1)$ 这个新定理，其中状态 s_1 是执行 unstack 动作后由 s_0 转换来的。

为了能推导出其他未受动作影响的状态成分，还需提供一组规则，称为画面公理，用以说明不受动作影响的成分。例如

$\text{ONTABLE}(z, s) \rightarrow \text{ONTABLE}(z, \text{do}(\text{unstack}(x, y), s))$ 说明 ONTABLE 关系不受 $\text{unstack}(x, y)$ 动作的影响；

$(\text{ON}(m, n, s) \wedge \text{NE}(m, x)) \rightarrow \text{ON}(m, n, \text{do}(\text{unstack}(x, y), s))$ 说明 ON 关系中的积木与 unstack 动作中的积木不相同时，该 ON 关系不会受影响；

$\text{COLOR}(x, C, s) \rightarrow \text{COLOR}(x, C, \text{do}(\text{unstack}(y, z), s))$ 说明积木颜色不受 $\text{unstack}(y, z)$ 动作的影响等等。

可以看出这种描述体系当问题状态的描述很复杂时，要求有大量的画面公理，而 STRIPS 系统的方法就不必要求有大量的显式画面公理。Green 系统中的画面公理已隐含在 STRIPS 形的规则中，即不出现在某一动作规则的删除表和添加公式中的谓词，就不受该动作的影响。当然执行动作后，要用其他办法来计算出完整状态的描述。

2. 正向产生式系统

机器人问题求解系统最简单的类型是一个产生式系统，即用

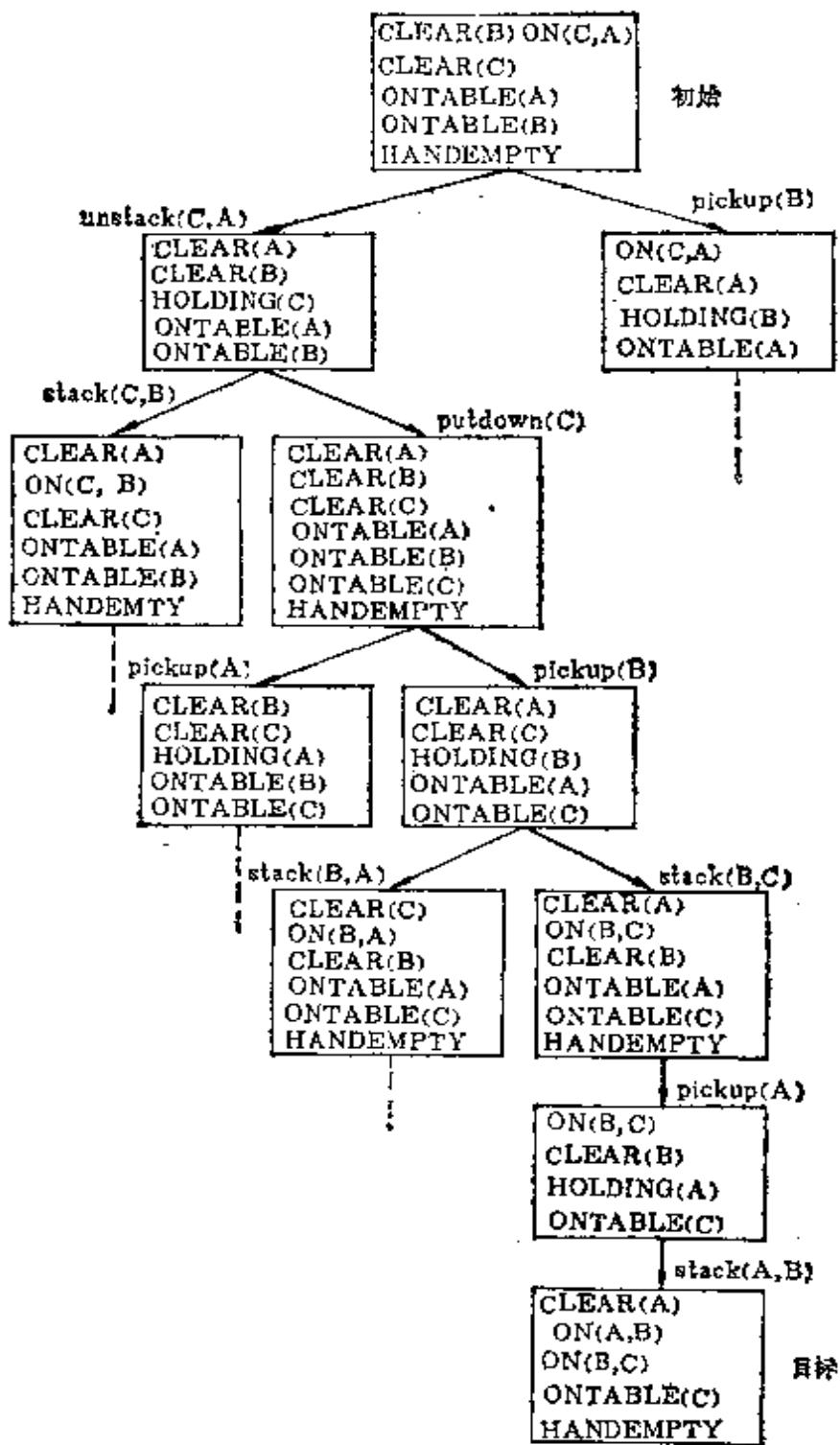


图 5.2 例子的部分搜索图

产生式系统可生成规划。我们用状态描述作为综合数据库，用STRIPS形的动作描述作为F规则，并采取某种搜索策略挑选应用规则，就可以找到到达目标状态的路径。图5.2给出图5.1的例子用正向产生式系统求解的结果。图5.3给出完整的状态空

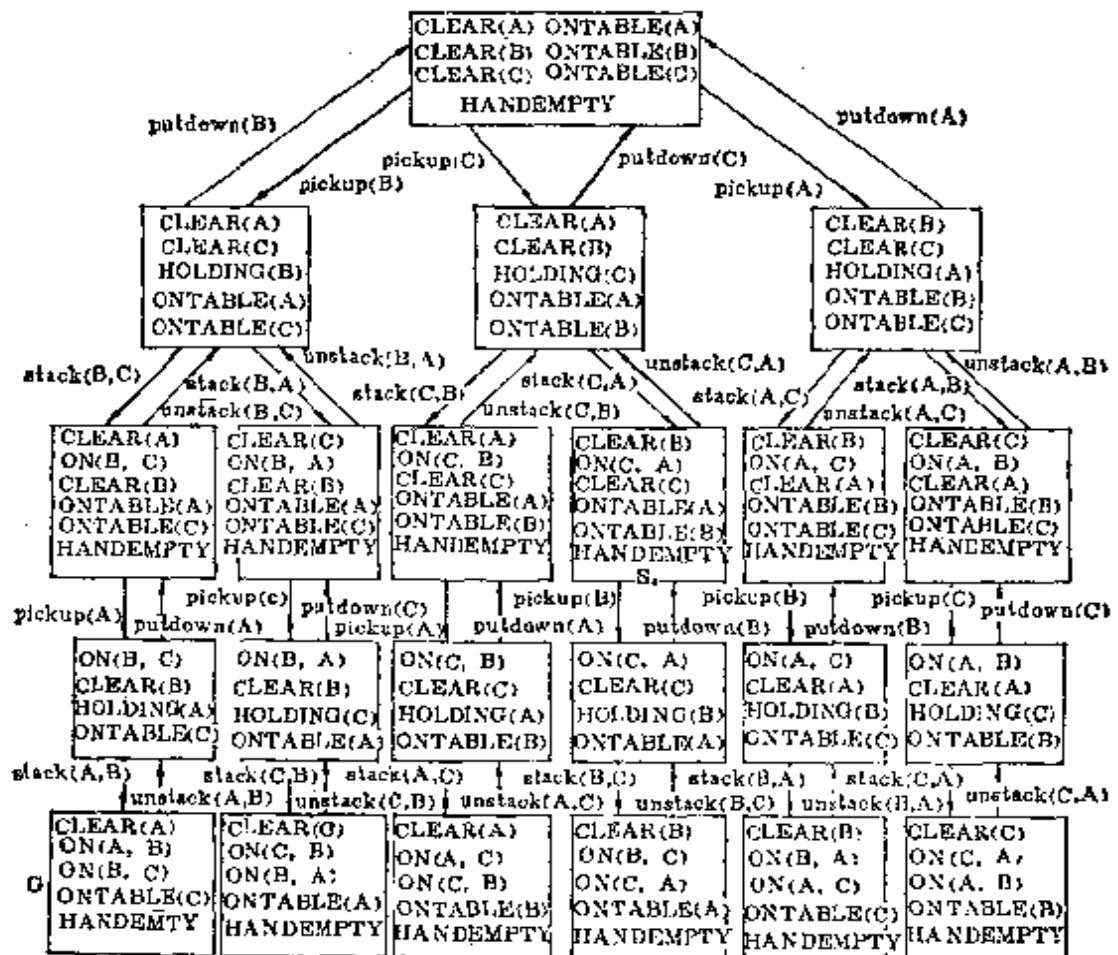


图 5.3 机器人问题的状态空间图

间图，并表示出动作的可逆性。得到的解序列是

{unstack(C,A), putdown(C), pickup(B), stack(B,C), pickup(A), stack(A,B)}

这也就是这个简单问题规划过程得到的机器人动作序列，通常称这个序列为达到目标的一个规划，可以看出这个规划的所有元素都是最基本的动作。对于复杂问题的规划结果，其元素可能是抽

象一级的目标，还需对这些目标作进一步的求解（或细化），才能得到最基本的动作序列。

5.3 规划的表示问题

上一节的简例中，已经给出了规划的结果。前面已指出过，机器人还要执行这个规划，也就是按照动作序列，一步一步地移动积木块，直至达到目标要求。当机器人能按预定的设想实现每一个动作的结果时，由这个序列给出的信息就已足够使用。实际上机器人在执行规划时，可能由于有一些动作未能达到预期的效果，如机械故障未能抓起积木，或者机械手移动过程意外地碰倒了别的积木等等，这时若再按步就班继续执行其他动作，肯定最后不能获得目标状态。因此规划执行过程需要监督，当动作执行后，偏离了预期效果，就应采取措施，避免盲目继续执行规划。一般有两种办法进行补救，一种办法是以当前的状态作为初始状态，重新进行规划，然后按新的规划执行；另一种办法是设法在规划结果中，除给出动作顺序关系外，还能表示出若干有用的附加信息，如一条 F 规则能为其他 F 规则提供那些先决条件，以及这些先决条件和规则之间的关系等。这样就可以利用这些信息，监视机器人执行规划，并在意外情况出现时能对规划进行修补。下面介绍一种三角形表的表示法，它能方便给出以上提到的这种相互有关系的附加信息，三角形表的一些特性对监视机器人规划的执行非常有用。

1. 三角形表的构造

图 5.4 表示上述例子规划的三角形表的结构，表中的每个单元登录规划中 F 规则的先决条件和加添部分，1 至 6 列的开头标记规划中相应 F 规则的名称，即第 j 列的 F 规则对应于规划动

作序列中第 j 个元素（动作规则名），第 0 列的内容是初始状态的各个分量。表中的最上面一行标记为第一行，若规划有 N 条规则，则最末一行 $i=N+1$ 。

							0
1	HANDEMPTY						
1	CLEAR(C)	1					
	ON(C, A)	unstack(C, A)					
2		HOLDING					
		(C)	2				
			putdown(C)				
3	ONTABLE		HANDEMPTY				
	(B)						
3	CLEAR(B)			3			
				pickup(B)			
4		CLEAR(C)	HOLDING				
		ONTABLE	(B)	4			
		(C)		stack(B, C)			
5	ONTABLE	CLEAR		HANDEMPTY			
	(A)	(A)					
					5		
					pickup(A)		
6			CLEAR	HOLDING			
			(B)	(A)	6		
					stack(A, B)		
7				ON(B, C)			
						ON(A, B)	

图 5.4 三角形表

三角形行列的编号确定后，表中各单元的内容具体规定如下：

(1) ($i < N + 1, j > 0$) 各单元的登录项是用了第 j 条 F 规则后，加到状态描述中的文字（即 A 的内容），这些文字要作为后续规则的先决条件或目标分量保存下来，故应分别填在后续规则或目标分量的相应行上。即第 j 条 F 规则以下单元的内容是其添加公式文字，可作为后续规则先决条件或目标分量，第 j 条 F 规则以左单元的内容正好是其先决条件公式文字。

(2) ($i < N + 1, 0$) 各单元是初始状态下，作为第 i 条 F 规则的先决条件保存下来的文字。

(3) ($i = N + 1, j$) 各单元（即 $N + 1$ 行）是初始状态描述的文字和 F 规则添加作为目标分量的文字，它们构成了整个目标状态。

此外，从三角形表还可给出所谓核的定义。第 i 个核是指表中第 i 行（包括该行）以下，第 i 列以左相交的部分内容，核内所有登记项是状态描述必须匹配的条件。若第 i 个核与当前状态匹配，则第 i 条规则及其后续规则组成动作序列可依次得到应用，直到达到目标。图 5.4 中双框线部分是第 4 个核，第 1 个核（0 列）包含每条 F 规则先决条件的初始状态分量和目标的初始状态分量，第 $N+1$ 个核则包含目标条件所有分量。

2. 三角形表的用法

三角形表中各个核具有监视机器人执行规划时所需要的信息，在规划执行开始阶段，整个规划对达到目标是可应用的，因为第 1 个核中的文字都能与初始状态描述相匹配（假定规划过程世界状态不发生变化）。于是就开始依次执行动作，这时假定系统的感知部分能根据客观世界变化的情况，不断修改状态描述，使其能与当前实际世界状态相一致，这样系统可以察看一下在一个动作执行之后，预期的核内容是否与状态描述匹配。例如执行完规划中的第 $(i-1)$ 个动作，马上察看一下第 i 个核中的文字是否与当前状态匹配，如果没有问题，则意味着规划剩余部分可以应用等等。实际上系统不仅仅只检查一下当前核文字匹配情况，而是可以从编号最大的一个核开始检查，这样可对执行时出现的问题进行有效的处理。例如从编号最大的核（表中最末一行，即目标核）依次往回检查每一个核内容，如果目标核被匹配上，那么不再继续执行规划，因此时已表明系统找到了目标（这种情况是可能出现的，如机械手运动过程意外碰倒了别的积木堆）；否则设检查出第 i 个核是当前最大的匹配核，那么系统下一步就应执行第 i 条 F 规则，执行完这条规则，再去搜索编号最大的核匹配情况等等；如果找不到匹配核时，就只能重新规划过程。由此看出，利用三角形表中核特性的信息，使规划执行中出现的意外情

况能得到灵活的处理，有时可省去执行一些不起作用的动作，有时则通过重复执行已执行过的动作，来克服执行中出现的失败，因而提高了规划执行的效率。现在再用前面的例子说明一下，设系统已执行了前面 4 个动作，情况正常，第 5 个核与当前状态匹配，于是执行第 5 个动作 $\text{pickup}(A)$ 。设这时系统误动作，执行了 $\text{pickup}(B)$ ，这时感知系统修改了状态描述，加上 $\text{HOLDING}(B)$ ，删去 $\text{ON}(B, C)$ ，而没有添加 $\text{HOLDING}(A)$ ，检查结果编号最大的匹配核变成 4（不出错时应为 6），因而系统的行为修改为再去执行第 4 个动作 $\text{stack}(B, C)$ ，而不是按原规定去执行第 6 个动作 $\text{stack}(A, B)$ 。

由于三角形表中各个核之间都有交集，因此可利用交的关系来找编号最大的匹配核。办法是从表的最末一行开始，从左向右扫描，寻找头一个与当前状态有一个不匹配文字的单元，若找不到这样的单元，则目标核被匹配。否则，若在第 i 列找到这个单元，则编号最大的匹配核不会大于 i ，这时可在第 i 列设置界限标记（列限），然后向上移一行，从左向右扫描到 $i-1$ 列就可以了，如又找到不匹配文字，则再设置一个列限，如此办法一直进行下去。若列限设置为 k ，扫描完第 k 行未出现不匹配的情况时，这时就找到了编号为 k 的匹配核，扫描过程结束。总之利用三角形表的表示法，可对规划执行过程进行监视和修正，从而解决了规划执行中产生的各种问题。

5.4 使用目标堆栈的简单规划方法

1. STRIPS 系统的工作过程

对于可将目标分解为若干子目标并可能有相互作用的这类问题，使用目标堆栈进行求解是早期的一种规划方法，STRIPS 系统就是用这种方法建立的求解系统。这个方法中的问题求解程序

是使用一个目标堆栈来存放子目标以及实现这些子目标所提出的一些算子（即STRIPS型规则），下面通过图5.5所示的例子来说明这种方法的基本思想。

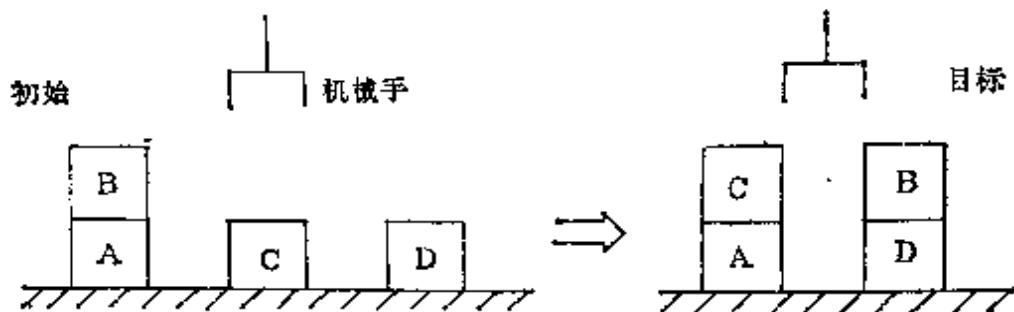


图 5.5 一个简单的积木世界问题

一开始目标堆栈只存放主目标

$ON(C, A) \wedge ON(B, D) \wedge ONT(AD)$

其中 $ONT(AD) = ONTABLE(A) \wedge ONTABLE(D)$

这个主目标可分解为4个子目标，每一个子目标对应于主目标的一个分量，对于子目标 $ONT(AD)$ 在初始状态下已达到，因此只需考虑其他两个未达到的子目标。有两种次序来分解这两个子目标，于是可产生两种可供选择的子问题，分别对应的目标堆栈如下：

(1) $ON(C, A)$

$ON(B, D)$

$ON(C, A) \wedge ON(B, D) \wedge ONT(AD)$

(2) $ON(B, D)$

$ON(C, A)$

$ON(C, A) \wedge ON(B, D) \wedge ONT(AD)$

STRIPS系统处理方法是把目标分解为子目标依次压入栈内后，首先是考察栈顶子目标的求解问题。若栈顶目标是一个子目标且与当前状态描述匹配时，则从栈顶删去（弹出），与此同时栈顶目标以下的各表达式都要应用该匹配置换例化；若栈顶目标是复合目标，则将复合目标中每一个子目标分量的文字按一定顺序添

在复合目标之上（可使用启发信息来排序），STRIPS 系统就按这个顺序从项目标开始依次对每个目标分量进行处理，求解完所有这些目标分量后，才轮到考虑这个复合目标。如果复合目标与当前状态描述匹配，则也删去，若不匹配，还得在栈顶再列出其目标分量，这种复合目标重新考虑主要是解决子目标相互作用所引起的问题。总之求解完一个目标分量就删去，在必要时删去的目标还可以重新考虑再次予以求解。

下面看一下这个具体例子的处理过程。两个子问题中，解决第 2 个目标堆栈比较简单，因此我们来讨论第 1 个目标堆栈。现在栈顶的子目标是单文字 $ON(C, A)$ ，当前状态下 $ON(C, A)$ 不为真，因此要找一条 F 规则能使其为真，这条规则的添加公式中应含有与文字 $ON(C, A)$ 相匹配的文字，如 $stack(x, y)$ ，然后用这条 F 规则匹配的例 $stack(C, A)$ 替代栈中的 $ON(C, A)$ ，在 F 规则 $stack(C, A)$ 之上，再加上它的先决条件公式 P 的匹配例 $CLEAR(A) \wedge HOLDING(C)$ 作为子目标，若 P 的例是一个复合目标，又不与当前状态匹配，那么其目标分量按一定顺序加到这个复合目标之上，这时可利用启发信息把 $HOLDING(C)$ 排到后面处理，因为 $HOLDING(x)$ 容易达到，但也容易破坏，一般后处理为宜，这样就得到新的目标堆栈

CLEAR(A)
HOLDING(C)
 $CLEAR(A) \wedge HOLDING(C)$
 $stack(C, A)$
 $ON(B, D)$
 $ON(C, A) \wedge ON(B, D) \wedge ONT(AD)$

接着检查 $CLEAR(A)$ 不为真，这时只有 $unstack(x, y)$ 这条 F 规则可使它为真，类似上面的过程又产生一个新目标堆栈

$ON(B, A)$

CLEAR(B)
HANDEMPTY
ON(B,A) \wedge CLEAR(B) \wedge HANDEMPTY
unstack(B,A)
HOLDING(C)
CLEAR(A) \wedge HOLDING(C)
stack(C,A)
ON(B,D)
ON(C,A) \wedge ON(B,D) \wedge ONT(AD)

这时检查 $ON(B, A)$ 为真, 被弹出堆栈, 接着 $CLEAR(B)$ 和 $HANDEMPTY$ 被相继弹出后, 栈顶变成复合目标。由于它是 F 规则 $unstack(B,A)$ 的先决条件公式, 且与当前状态匹配, 故也被弹出, 因而栈顶的 F 规则 $unstack(B,A)$ 可以应用, 于是被应用到当前状态描述上, 产生一个新的状态描述代替原来的状态描述, 而且该规则也被删去, 系统应记住这条用过的规则, 以便最后构造解序列。

接下去问题求解程序就要以这个新状态
 $ONT(ACD) \wedge HOLDING(B) \wedge CLEAR(A)$
和目标堆栈

HOLDING(C)
CLEAR(A) \wedge HOLDING(C)
stack(C,A)
ON(B,D)
ON(C,A) \wedge ON(B,D) \wedge ONT(AD)

出发进行求解, 要达到目标 $HOLDING(C)$ 有两条路径, 同样可利用启发信息选 $pickup(C)$ 这个方案, 因选 $unstack(C,x)$ 时, 还得先执行 $stack(C,x)$, 这样目标堆栈为

$CLEAR(x) \wedge HOLDING(C)$

```

stack(C, x)
CLEAR(C)
HANDEMPTY
ON(C, x) ∧ CLEAR(C) ∧ HANDEMPTY
unstack(C, x)
CLEAR(A) ∧ HOLDING(C)
stack(C, A)
ON(B, D)
ON(C, A) ∧ ON(B, D) ∧ ONT(AD)

```

看出为解决 HOLDING(C)，结果栈顶又出现 HOLDING(C)，即又回到原始子目标且增加了新的子目标，这是不可取的。

类似的做法进行下去最后可得到解序列

{unstack(B, A), stack(B, D), pickup(C), stack(C, A)}

我们可以把 STRIPS 系统的求解过程看成是一个产生式系统，该产生式系统的综合数据库是由当前状态和目标堆栈组合构

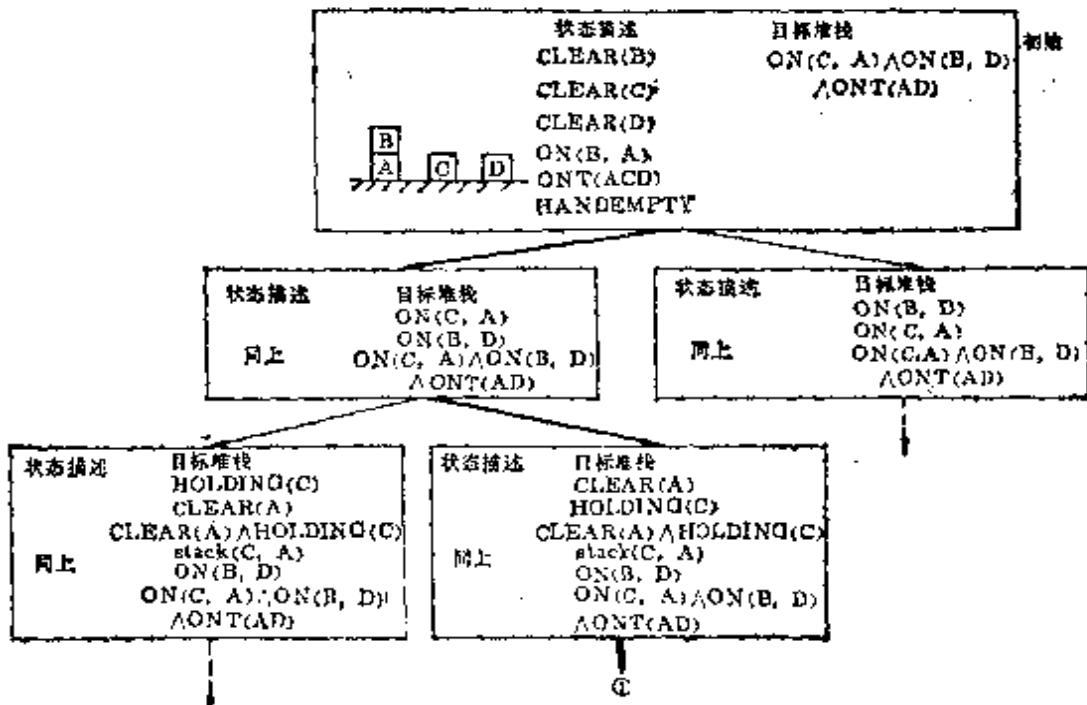


图 5.6 STRIPS 系统的搜索图

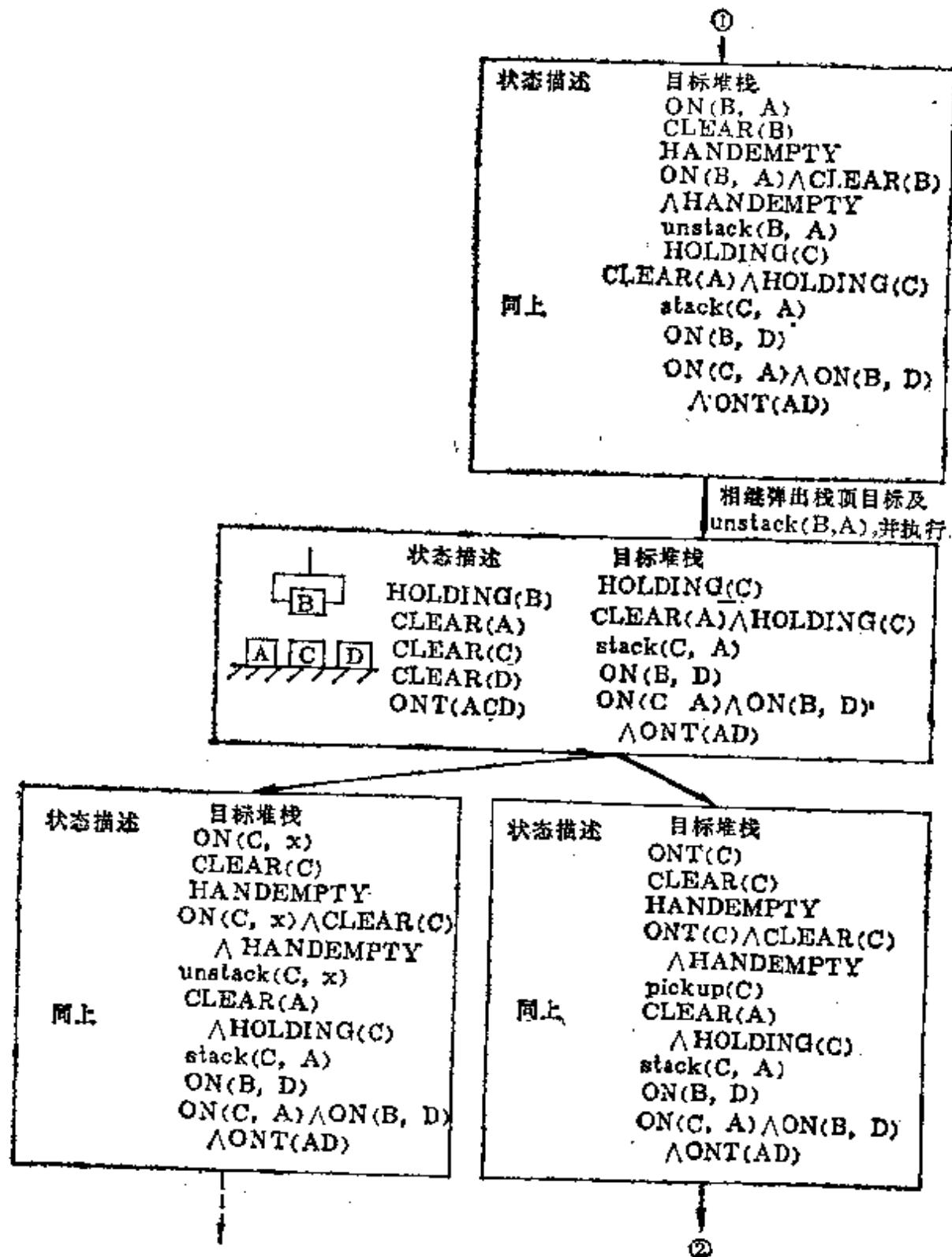


图 5.6 STRIPS系统的搜索图

成，产生式规则是处理顶层目标的规则，从目标堆栈中选出 STRIPS F 规则，作用于当前状态，

以便改变状态和目标堆栈这两个组成部分的描述。这里要注意与 STRIPS F 规则的区别，后者只是相当于机器人动作的模型。图 5.6 表示出这个产生式系统搜索过程的部分搜索图，终节点是目标堆栈为空的那个综合数据库构成。从初始节点到终节点的路径可得出如上所示的解序列。

下面再来看一看图 5.1 所示的例题用 STRIPS 系统求解时会出现的问题。对目标栈来说，开始求解时也有两种选择：

- (1) $ON(A, B)$
 $ON(B, C)$
 $ON(A, B) \wedge ON(B, C)$
- (2) $ON(B, C)$
 $ON(A, B)$
 $ON(A, B) \wedge ON(B, C)$

设系统先选择第一种的目标堆栈开始求解，那么先考虑如何实现子目标 $ON(A, B)$ ，根据上面的方法进行处理，最后可得目标堆

栈为

$CLEAR(C)$

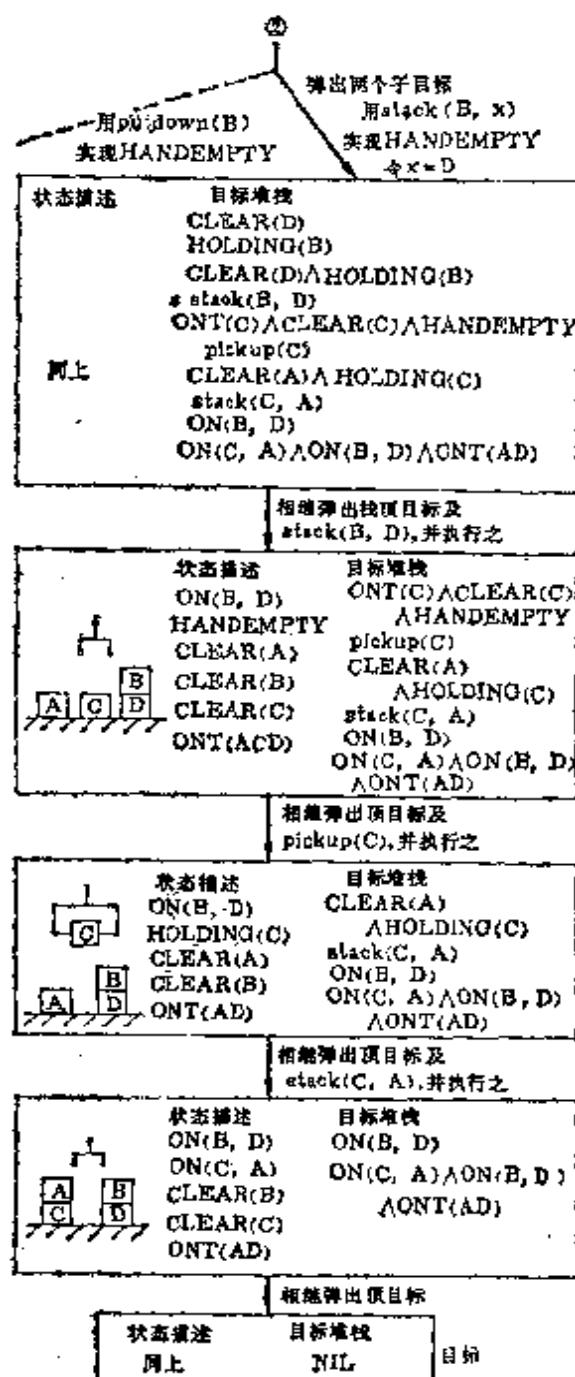


图 5.6 STRIPS 系统的搜索图

```

HANDEMPTY
CLEAR(C) ∧ HANDEMPTY
unstack(C, A)
HANDEMPTY
CLEAR(A) ∧ HANDEMPTY
pickup(A)
CLEAR(B) ∧ HOLDING(A)
stack(A, B)
ON(B, C)
ON(A, B) ∧ ON(B, C)

```

接着相继将匹配的子目标一一删去，直到 $\text{pickup}(A)$ 的前提条件 HANDEMPTY 。然后再调用 $\text{putdown}(C)$ 进行处理，直至实现 $\text{ON}(A, B)$ ，这时目标堆栈为

```

ON(B, C)
ON(A, B) ∧ ON(B, C)

```

并已应用了 4 条 F 规则 $\{\text{unstack}(C, A), \text{putdown}(C), \text{pickup}(A), \text{stack}(A, B)\}$ ，相应的新状态为：

```

ON(A, B) ∧ CLEAR(C) ∧ CLEAR(A) ∧ HANDEMPTY
∧ ONT(BC)

```

现在考虑实现 $\text{ON}(B, C)$ ，可用 $\text{stack}(B, C)$ ，为此又需执行 $\text{unstack}(A, B)$ ，即要把上面实现的 $\text{ON}(A, B)$ 破坏掉，这就是子目标间相互作用的问题，系统必须再次把 $\text{ON}(A, B)$ 放入目标栈。这个问题最后得到的完整规划是

```

{unstack(C, A), putdown(C), pickup(A), stack(A, B),
unstack(A, B), putdown(A), pickup(B), stack(B, C), pic-
kup(A), stack(A, B)}

```

执行它可以达到目标，但其中 3~6 这 4 条 F 规则是走弯路产生的结果。同样采用方案 2，情况也不会得到改善，原因在于子目

标间有相互影响。因此象这样的问题 STRIPS 系统就不是很有效。

改进走弯路的办法有两种。一是用某种方法来修正一下已经得到的规划，例如对上例的规划，先寻找其中互相冲突的操作对，然后删掉（如 $stack(A, B)$ 和 $unstack(A, B)$ ， $pickup(A)$ 和 $putdown(A)$ ），这样就将无效的 F 规则丢弃，改善了所得结果。这种补救办法虽能去掉一些无用动作，但对复杂问题，有时寻找冲突的操作对不太容易，而且求解过程耗费很多无效的工作量，总的效率是不能令人满意的。另一种是改进规划方法，这将在下一节中再讨论。

最后还应指出，有些问题 STRIPS 系统不可能求到解，有关论述可参阅参考文献。

2. STRIPS 的控制策略

STRIPS 系统的控制策略主要要考虑以下几个问题：

（1）必须决定目标堆栈中复合目标的各分量以怎样的顺序添加到复合目标上面。

一般可先找出所有与当前状态描述相匹配的那些分量，放在栈顶并立即删去，然后对未匹配上的子目标分量排序。这样可得到不同组合排序的若干个目标堆栈的节点，要选一个节点进行求解。

（2）选择处理目标分量的规则

（3）在目标堆栈中有存在量词量化的变量出现时，就可以建立起若干不同匹配例的后继节点，要从中选一个匹配置换例进行求解。

图搜索策略可应用来解决这些问题，在选择求解节点的过程中，可以利用一些启发信息。如考虑目标堆栈的长度，目标堆栈中子目标实现的困难程度，以及使用 STRIPS F 规则的代价等，来建立评价函数。

也可用回溯策略来求解。设初始状态描述用一个全程变量 S 标记, STRIPS 系统要达到的目标用变元 G 标记, 则 STRIPS 的程序类似如下递归过程:

递归过程 STRIPS(G)

- ① Until S 匹配 G, do:
- ② begin
- ③ $g :=$ 不匹配 S 的一个 G 的分量, 一个不确定的选择,因而是一个回溯点。
- ④ $f :=$ 一条 F 规则, 其加添公式中包含有一个与 g 匹配的文字, 另一个回溯点。
- ⑤ $p := f$ 的合适匹配例的先决条件公式;
- ⑥ STRIPS(p); 求解子问题的递归调用。
- ⑦ $S := f$ 的合适匹配例应用于 S 得到的新状态;
- ⑧ end

3. 手段-目的分析 (means-ends analysis) 和 GPS (General Problem Solver)

GPS 是 60 年代建立的一个一般问题求解系统, 使用一种叫做手段-目的分析的方法进行求解。它和 70 年代建立的 STRIPS 系统所采用的方法类似。已知状态 S 和目标 G, 及一组关键性的 F 规则 (这些规则可消除 S 和 G 之间的差别)。S 和 G 之间差别的计算是由一个与应用领域有关的函数来实现。此外每一个应用领域都要提供一个差别表, 来确定 F 规则与差别之间的连系。当 F 规则的先决条件都满足时, 这条 F 规则就可应用于当前状态描述上。GPS 的程序也类似于如下递归过程。

递归过程 GPS(G)

- ① Until S 匹配 G, do:
- ② begin

- ③ $d := S$ 和 G 之间的一个差别；一个回溯点。
- ④ $f :=$ 与减少 d 有关的一条 F 规则；另一个回溯点。
- ⑤ $p := f$ 的合适匹配例的先决条件公式；
- ⑥ $GPS(p)$ ；求解子问题的递归调用。
- ⑦ $S := f$ 的合适匹配例应用于 S 得到的新状态；
- ⑧ end

$GPS(G)$ 中找出差别并为减少差别选择 F 规则的过程叫做手段-目的分析。STRIPS(G) 可认为是 $GPS(G)$ 的特例， GPS 中的 S 和 G 的差别就是没有与 S 匹配上的那些 G 的分量，加添表中包含有一个 L 文字的所有 F 规则都被认为是与减少差别 L 有关的规则。

原始的 GPS 系统都是递归方式工作的，当然也可建立具有图搜索控制方式的 GPS 系统。

5.5 用目标集的非线性规划方法

上一节讨论了用目标堆栈制定规划的方法，它可用于求解有若干个子目标组成复合目标这类问题，步骤是先求解一个子目标，再求解另一个，直到全部子目标都达到为止。用这种方法生成的规划是由实现各个子目标的动作序列依次连接组成，即规划是{（实现第一个子目标的子规划）+（实现第二个子目标的子规划）+…}。很容易想像得到，有些问题子规划之间的动作若能交叉执行（例如某个子规划执行若干步之后，便转去执行另一子规划，然后再回过头来继续执行），则生成的规划会更合理和有效。例如图 5.1 的例子，首先实现 $ON(A, B)$ ； $unstack(C, A)$ ， $putdown(C)$ ，接着转去实现 $ON(B, C)$ ； $pickup(B)$ ， $stack(B, C)$ ，最后再完成 $ON(A, B)$ ； $pickup(A)$ ， $stack(A, B)$ 。由于这样组成的规划不是完整子规划的线性序列，所以称

这类规划为非线性规划。

一种寻找这类规划的方法是把子目标看成一个集合，使用逆向推理的方式进行求解，而不是像 STRIPS 系统那样，把子目标排成栈队列，只能先处理栈顶元素。因而求解过程就可以根据具体情况，每次可从集合中选出任意一个元素进行处理，从而能实现非线性规划过程。下面还以图 5.1 的例子，用逆向产生式系统求解，来说明这种规划方法的基本思想。

1. 逆向产生式系统

产生式系统的综合数据库由目标描述（文字的合取式）组成，它表示成集合的形式，产生式规则集由一组 B 规则（逆向应用相应的 F 规则）来建立。系统从目标表达式出发，不断应用 B 规则来产生子目标描述，当产生的子目标描述与初始状态描述的事实相匹配时，系统就成功结束。现在用图搜索策略求解图 5.1 的例子，得到的部分搜索树如图 5.7 所示。初始节点①的目标表达式集是 {ON(A, B), ON(B, C)}，它可以生成两个子目标节点②和③，因为是逆向推理，所以有两个可供选择的 B 规则（②和③的状态相当于规则中最后一条 F 规则执行时的当前状态），对节点②可用 stack(A, B)，对节点③可用 stack(B, C)。如果选用 stack(B, C)，则其前提条件公式 CLEAR(C) 和 HOLDING(B) 以及文字 ON(A, B) 都必须为真，但假定机械手一次只抓取一个积木，那么 ON(A, B) 就无法去抓 B，这是矛盾的（系统中这个矛盾可用一组公理来发现），因此含有这个矛盾节点的路径应删去，只剩下节点②要考虑。

和上面的推理过程差不多，节点②的三个子目标将生成 4 个子子目标节点④—⑦，对应的 B 规则为：unstack(x, B), pickup(A), unstack(A, x), stack(B, C)。可以看出节点④ HANDEMPTY 和 HOLDING(A) 有矛盾；节点⑥中当 $x=C$ 时，

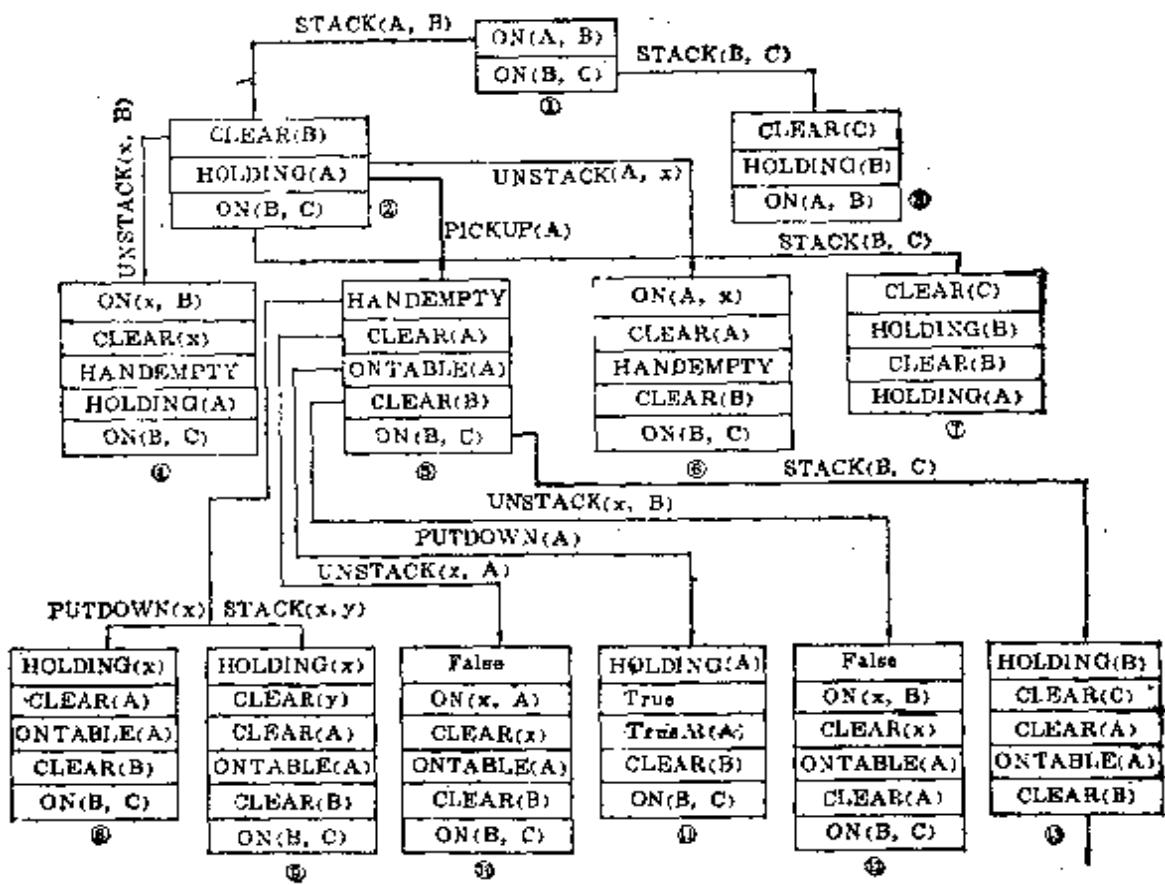


图 5.7 逆向系统的部分搜索树

$ON(A, C)$ 与 $ON(B, C)$ 有矛盾, 而 $x=B$ 时, 得到一个含有初始目标的扩大集, 这不可取; 节点 ⑦ $HOLDING(B)$ 和 $HOLDING(A)$ 有矛盾, 因此最后只剩下节点 ⑤ 可进一步生成。

概括一下, 到目前为止逆向系统生成子目标的过程是:

设节点的目标集中有一个文字 L , 它与某一条 F 规则加添公式中的一个文字 L' 合一 (mg 为 u), 则逆向应用这条规则 (称 B 规则), 把其先决条件公式的例 P_u 作为子目标分量替代 L , 目标集中其他的文字被保留在子目标集中, 当这些非 L 的文字在 F 规则执行前和执行后均为真时, 这样做不会有什么影响。但当 F 规则执行动作后, 非 L 文字的这些目标分量中有一个受影

响并使其不再为真时会出现什么情况呢？显然这时不能直接把这些文字放进子目标集，而必须对它们进行所谓的回归（Regression）过程，下面先讨论回归过程，然后再来继续例子的搜索。

2. 目标回归过程

回归可看作是 F 规则的逆向应用，通过一条 F 规则回归一个目标，其目的是要决定在执行动作之前，什么状态必须为真才能使执行动作后该目标仍为真。如用 STRIPS F 规则的例回归目标文字时，可形式描述如下：

令 $\text{Regression}(Q, Fu)$ 表示文字 Q 通过一条 F 规则基例 Fu 的回归，并简记为 $R(Q, Fu)$ ，则

```
IF  $Qu \in Au$  THEN  $R(Q, Fu) = \text{True}$ 
ELSE IF  $Qu \in Du$  THEN  $R(Q, Fu) = \text{False}$ 
ELSE  $R(Q, Fu) = Qu$ 
```

例如 $R(\text{ON}(A, B), \text{pickup}(C)) = \text{ON}(A, B)$ ，因目标文字 $\text{ON}(A, B)$ 不是这条规则 P 中或 D 中的文字，故 $\text{ON}(A, B)$ 的回归就是其本身。

又如 $R(\text{ON}(A, B), \text{stack}(A, B)) = \text{True}$ ，这个目标文字 $\text{ON}(A, B)$ 可通过该规则实现（规则的先决条件当然必须为真），故回归为真。

再如 $R(\text{HANDEMPTY}, \text{pickup}(A)) = \text{False}$ ，执行这条规则，破坏了 HANDEMPTY ，故回归为假。

总之目标回归过程是通过应用 B 规则来生成子目标集的，设目标为 $G = L \wedge G_1 \wedge \dots \wedge G_N$ ，逆向应用 F 规则生成得到子目标 $G' = P_u \wedge G'_1 \wedge \dots \wedge G'_N$ 时，则其中每一个 G'_i 称为 G_i 的回归（通过 F 规则的例），即 $G'_i = \text{Regression}(G_i, Fu)$ ，从 G_i 求得 G'_i 的过程称为回归过程。

3. 回归过程的具体应用

有时候用一条规则实现一个目标文字时，另外一些目标也可能被回归为真，例如用 $\text{stack}(B, D)$ 实现目标 HANDEMPTY ，这时另一个目标 $\text{ON}(B, D)$ 也回归为真，因此只要一个目标一旦回归为真，就可从目标集中删掉，这是由于不论规划执行前什么状态为真，这个目标都为真。另外当一个子目标集中含有 False 时，显然这个子目标节点就不可实现，以后的路径就不必继续搜索。

现在回到图5.7中节点⑤，看一看怎样应用回归过程。先考虑第一个目标分量 HANDEMPTY ，有两条规则可实现： $\text{putdown}(x)$ 和 $\text{stack}(x, y)$ ，但要求这两条规则的先决条件 $\text{HOLDING}(x)$ 为真，显然其中变量 x 约束为任一积木都会产生矛盾，所以相应生成的节点⑧和⑨均应删除。接着第二个目标分量 $\text{CLEAR}(A)$ ，只有 $\text{unstack}(x, A)$ 可用，但此时第一个分量 HANDEMPTY 回归为假，因此节点⑩的路径应删除。第三个目标分量 $\text{ONTABLE}(A)$ 可用 $\text{putdown}(A)$ 实现，生成节点⑪，实际上如果利用一些启发信息，这个节点可不必考虑，因为它在初始状态就为真，且不受整个过程规划动作的影响（处理 HANDEMPTY 时，虽初始状态为真，但其受动作影响，故不能不考虑）。再看目标分量 $\text{CLEAR}(B)$ ，可用 $\text{unstack}(x, B)$ 实现，但 HANDEMPTY 回归为假，所以节点⑫也要抛弃。最后 $\text{ON}(B, C)$ 可用 $\text{stack}(B, C)$ 实现，生成得到节点⑬。由此看出通过应用 B 规则和回归过程，搜索时可实现有效的剪枝，如果继续从节点⑬搜索下去，最终将找到一个目标集，其全部元素完全与初始状态描述匹配。图 5.8 给出其余部分的搜索图。根据从目标到事实的解路径逆向排列 F 规则，可得规划为

{ $\text{unstack}(C, A)$, $\text{putdown}(C)$, $\text{pickup}(B)$, $\text{stack}(B, C)$,

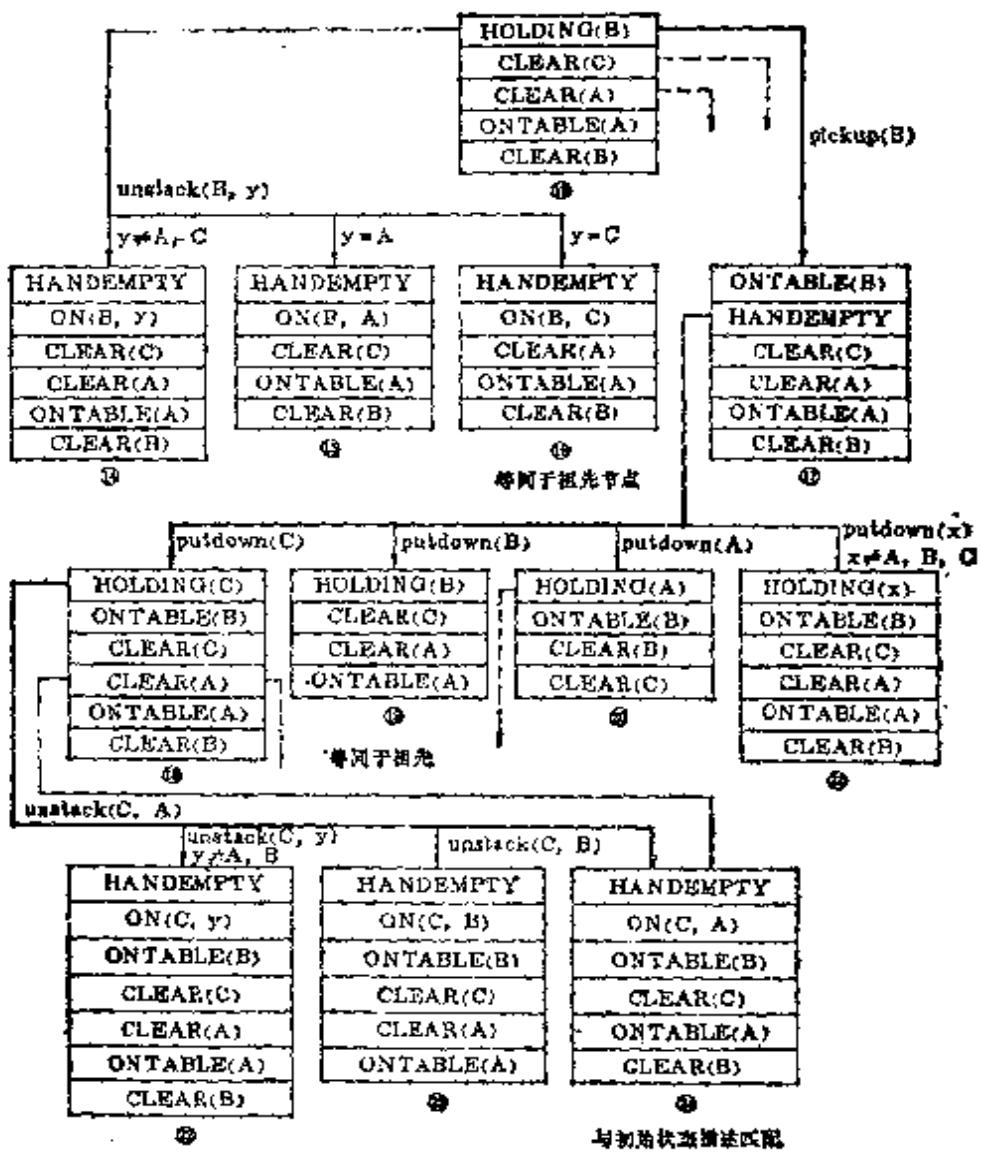


图 5.8 逆向系统后半部的搜索图

$\text{pickup}(A), \text{stack}(A, B)\}$

最后指出，对于未完全例化的 F 规则回归过程，处理起来稍微复杂一些，本节不再详细举例。这里强调一点，非线性规划方法中，主要是把目标表达式用集合表示，因此在搜索过程一切可能的子目标排列都考虑到，因而容易构造出非线性的规划来。这个方法的缺点是当问题规模较大时，特别是子目标之间并无相互作用，那么这种方法灵活性的优点得不到发挥，反而不能有

效的剪枝，致使搜索空间十分庞大。另一点是这个方法不能区分重要目标和一般要求的目标之间的关系，所以有可能会陷入某一个非主要目标细节的求解，最后才发现这个规划不适用。因此需要在规划过程先作出满足主要目标的规划，然后再进行规划的细化，这就是分层规划方法的思想，下面将作进一步讨论。

5.6 分层规划方法

前面讨论的生成达到目标的各种规划方法都是在一个层次内进行求解，也就是说在整个问题空间中来寻找能满足要求的规划。例如用逆向系统求解时，是从目标开始逆向推理，先考察达到这一目标的条件，并把这些条件作为子目标进行求解，然后再考察达到这些子目标的条件，如此一步一步地搜索下去。实际上在求解实际问题中，往往可以把某一个目标和子目标的条件仅仅当成细节看待，暂时搁在一边，而把精力集中到先根据主要的目标条件，确定规划的基本步骤，然后再处理细节问题，并将其结果补充到基本规划中去。这就是分层规划的思想，把提出的目标条件和达到这些目标条件的规则分层加以组织，最细致的条件和基本的动作在最低层规划中考虑，最主要的条件和规则放在最高层先处理。实际上这就是把问题空间划分为若干个不同抽象程度的空间，在最一般的抽象空间中作最高层的规划，在具体实际问题空间中作最低层的规划。这样以分层的方式，可使每一层的规划都具有合理的长度，使得问题求解的复杂程度大为降低，这种求解策略称为分层规划 (Hierarchical planning)。

分层规划是通过逐层都构造一个规划的方法来实现的，在每一层中做规划时，都可以使用任何一种单层规划的方法，这时只考虑主要条件，有些条件被当成细节（即在该层中不起主导作用），留到后面的层次去考虑。当这些细节在某一低层中作用变

得明显时，还必须有办法在高层规划中增补实现这些细节的部分。下面将通过几个系统，简要讨论一下分层规划的方法。

1. ABSTRIPS系统 (1974)

ABSTRIPS系统是STRIPS系统的改进，它引入了分层规划的思想。ABSTRIPS系统求解问题的方法比较简单，这里用图5.1的初始状态，实现目标 $ON(C, B) \wedge ON(A, C)$ 来说明这个方法。

系统中所使用的规则仍然是STRIPS型的F规则。为了能区别先决条件主次关系，以便能推迟处理一些次要条件，实现分层处理，因此必须规定各种条件（包括目标条件）的级别，例如规定 $ON(x, y)$ 具有最高的关键值，而 $HANDEMPTY$ 容易达到，赋予最低的关键值，这些值反映出满足先决条件的困难程度。该例中，每一个条件属于那一层就用这个关键值来表示，大的关键值属高层，小的则属低层。设ABSTRIPS F规则的条件分三层处理，则可表示如下：

(1) $pickup(x)$

P 和 D : $ONTABLE(x)$, $CLEAR(x)$, $HANDEMPTY$
 A : $HOLDING(x)$

(2) $putdown(x)$

P 和 D : $HOLDING(x)$
 A : $ONTABLE(x)$, $CLEAR(x)$, $HANDEMPTY$

(3) $stack(x, y)$

P 和 D : $HOLDING(x)$, $CLEAR(y)$
 A : $HANDEMPTY$, $ON(x, y)$, $CLEAR(x)$

(4) $unstack(x, y)$

1 2 3
P 和 D: HANDEMPTY, CLEAR(x), ON(x, y)
A: HOLDING(x), CLEAR(y)

这里规定的键值只标记在 P 和 D 的文字上面, A 中的文字没有标记, 这样当应用一条 F 规则时, 加添公式的所有文字不分层处理, 都被加到状态描述中。由于键值分三个等级, 因此 ABSTRIPS 分三层求规划。

最高层只考虑最高键值的条件, 即键值为 3 的那些条件, 值低于 3 的条件当作细节, 在这一层中不予考虑。由于主目标 $ON(C, B) \wedge ON(A, C)$ 实际上是两个键值为 3 的条件, ABSTRIPS 可任选一个来处理, 设现在先处理 $ON(C, B)$, 这样就可把 $stack(C, B)$ 加到目标堆栈中, 而其先决条件的键值均低于 3, 所以不必加到堆栈上。这时栈顶没有其他条件, 所以可应用 F 规则 $stack(C, B)$ 并得到新状态描述, 这里要注意其中有应被删除的项, 也因键值小于 3 而未删去, 这可能引起矛盾的状态描述, 但不会对规划产生影响。这样进行下去, 可得这一层的规划过程搜索图如图 5.9 所示, 求得的解序列是 $\{stack(C, B), stack(A, C)\}$, 这就是达到目标的高层规划。从高层的含义来看, 可认为摆积木的操作是最重要的, 规划过程要优先考虑, 接着在低层规划中再考虑加入细节。

生成出最高层的规划后, 就把所得规划 $\{stack(C, B), stack(A, C)\}$ 传给下一层。这一层考虑键值为 2 或高于 2 的那些条件, 以便开始考虑某些细节。这时目标堆栈应包括上一层求得的 F 规则序列以及这个序列中各 F 规则满足键值的先决条件, 这样就把上一层的规划有效地传递给下一层, 再把主目标放在目标堆栈底部, 这样构造初始目标堆栈为

HOLDING(C) \wedge CLEAR(B)
 $stack(C, B)$

HOLDING(A) \wedge CLEAR(C)

stack(A, C)

ON(C, B) \wedge ON(A, C)

状态描述	目标堆栈
CLEAR(B)	ON(C, B) \wedge
CLEAR(C)	ON(A, C)
ON(C, A)	
HANDEMPTY	
ONTABLE(A)	
ONTABLE(B)	

状态描述	目标堆栈
CLEAR(B)	ON(A, C)
CLEAR(C)	ON(C, B) \wedge
ON(C, A)	ON(A, C)
HANDEMPTY	
ONTABLE(A)	
ONTABLE(B)	
ON(C, B)	

状态描述	目标堆栈
CLEAR(B)	ON(C, B)
CLEAR(C)	ON(A, C)
ON(C, A)	ON(C, B) \wedge
HANDEMPTY	ON(A, C)
ONTABLE(A)	
ONTABLE(B)	

状态描述	目标堆栈
CLEAR(B)	stack(A, C)
CLEAR(C)	ON(C, B) \wedge
ON(C, A)	ON(A, C)
HANDEMPTY	
ONTABLE(A)	
ONTABLE(B)	
ON(C, B)	

状态描述	目标堆栈
CLEAR(B)	stack(C, B)
CLEAR(C)	ON(A, C)
ON(C, A)	ON(C, B) \wedge
HANDEMPTY	ON(A, C)
ONTABLE(A)	
ONTABLE(B)	

状态描述	目标堆栈
CLEAR(B)	CLEAR(A)
CLEAR(C)	NIL
ON(C, A)	
HANDEMPTY	
ONTABLE(A)	
ONTABLE(B)	
ON(C, B)	
ON(A, C)	

图 5.9 第一层规划过程

从这个初始目标堆栈出发，和最高层的做法类似，可以找到一个可能的解序列 {unstack(C, A), stack(C, B), pickup(A), stack(A, C)}。如果找到解，ABSTRIPS 会返回到上一层去寻找另外的解，然后再传下来继续规划过程。可以看出上一层传下

来的规划有效地约束了这一层的搜索，并且很容易增补一些达到细节条件的 F 规则，因而提高了搜索效率，减少了组合爆炸。

在最低层开始处理关键值为 1 的条件，这一层初始目标堆栈包括第二层求得的 F 规则序列及这些规则的先决条件（最低层自然就是全部条件）。对这个例子，规划结果只是证实了第二层的解就是该问题最细致一层的正确解。

从这个例子看出，ABSTRIPS完成分层规划的方法比较简单，只用了赋关键值的方法实现对先决条件公式和删除表中文字重要性的分级。对于更复杂的问题，ABSTRIPS 系统要比单层规划的 STRIPS 系统更为有效。此外对分层中问题求解这一具体问题可以采取不同的方法，即每一层用的基本问题求解程序不一定 是 STRIPS 的方法，只要高一层产生的解可用来约束和引导这一层的任何求解方法都可使用。

2. NOAH系统 (1977)

NOAH 系统是 70 年代中研究的一个分层规划系统，规划的生成过程是采用一种过程网来描述。一个过程网是由若干节点以及它们之间的连接弧组成，节点有三类：表示过程信息（算子或规则）的节点；表示陈述信息（状态描述）的节点；表示分支(S)和合并 (J) 的节点（是一种虚节点）。连接弧表示出节点间某种并行或串行的顺序关系，系统采用分层规划的方法，先构造一个抽象的规划轮廓，然后再逐步补充进愈来愈多的细节。

该系统采用最少许诺策略 (Least Commitment strategy) 来选择算子（规则）执行的顺序，这是一种不必考虑算子序列的所有排列顺序来求非线性规划的方法，下面通过 NOAH 系统求解图 5.1 的例子进一步说明这种分层规划的思想。

图 5.10 给出 NOAH 系统求解积木问题非线性规划的过程，图中方框所示的节点标记已经被选定加入规划中的算子，带圆弧

的节点则标记尚未满足的目标。这个例子中所用的算子和前几节所使用的 F 规则稍有差别，例如 stack 的操作表示把任意一个物体放在另外的任一物体之上，只要这两个物体都是空顶，而且还包括拣起要移动的物体这个操作。

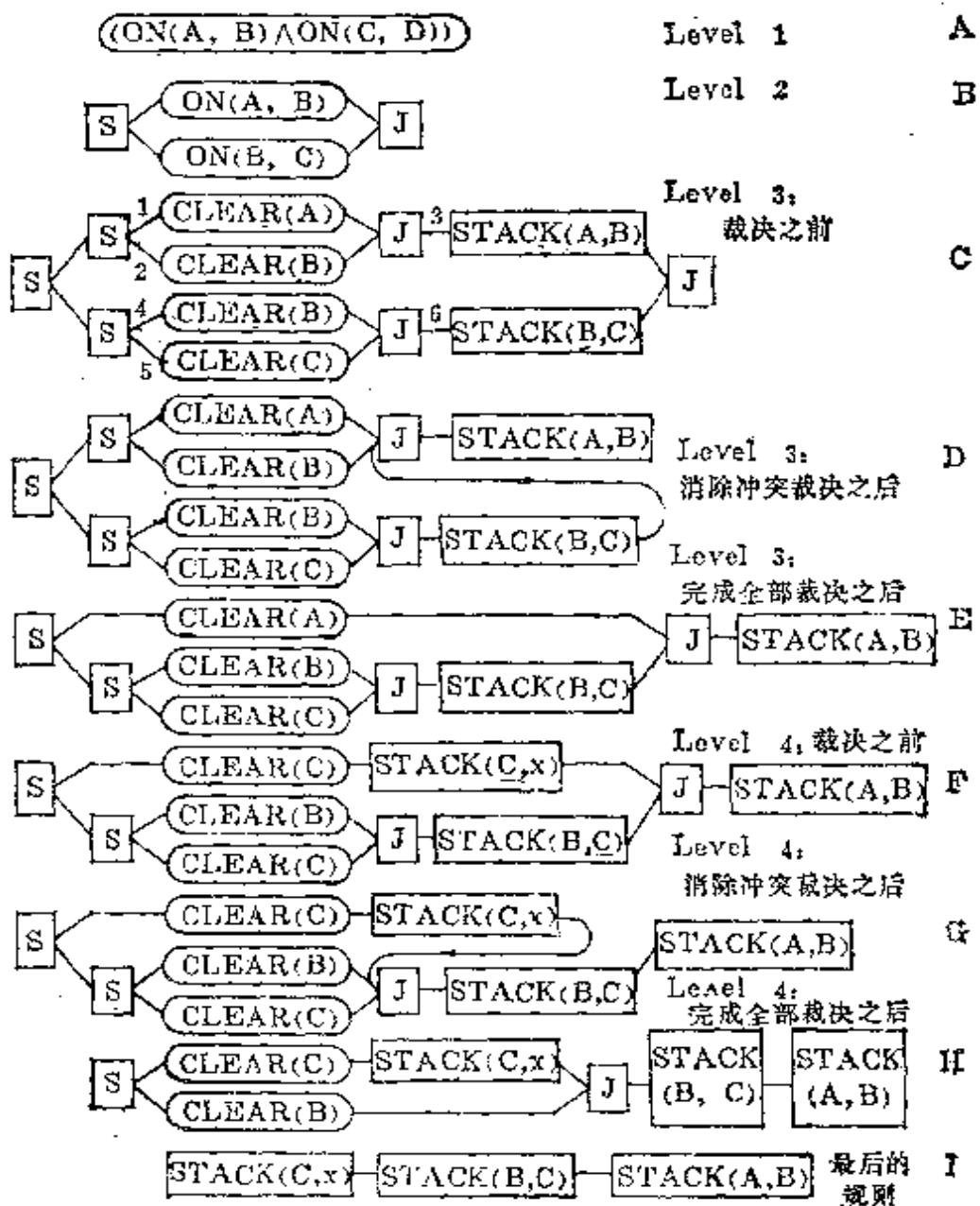


图 5.10 NOAH 系统求积木问题的非线性规划

问题求解程序的初始状态如图 5.10 中图 A 所示，首先把问

题分解为两个子问题(图B),这时问题求解程序决定用stack算子来实现每一个目标,但是还没有考虑该算子的先决条件。S节点表示规划分叉,并行的两个分量都必须实现,但其先后次序尚未确定。J节点表示两个分支重新汇合。再下一层就要考虑stack的先决条件,它们所对应的两个积木为空顶,于是系统记下stack操作可以执行时所必须满足的条件(图C),图C实际上表达了两个算子之间两种定序关系:一种是必须满足的排序关系,即清顶操作必须在stack操作之前;另一种是暂不考虑排序关系,即stack(A,B)和stack(B,C)那一个先操作,当前不必急于确定。

NOAH系统还采用一组判据(critics)来考察规划和检测子规划之间的相互作用,每一个判据是一段子程序,可对提出的规划进行具体的观察,这种判据的概念已用于各种规划生成系统。例如 HACKER系统(1975),判据只用来拒绝接受不可满足的规划,而NOAH系统中,判据被用来提出修改规划的各种方法。例如第一个调用的判据是消除冲突的判据,它首先构造一张表,列出规划中不只出现一次的所有文字,这个表格有如下条目:

CLEAR(B): 断言: 节点2 "clear B"

否定: 节点3 "stack A on B"

断言: 节点4 "clear B"

CLEAR(C): 断言: 节点5 "clear C"

否定: 节点6 "stack B on C"

当某一个已知的文字在一种操作执行前必须为真,但又受到另一个操作的限制又不能实现时,两个操作之间的定序约束就出现,在这种情况下,显然要求文字为真的那个操作必须先执行。从表中看出,CLEAR(B)对stack(B,C)操作来说必须为真,但它又会被stack(A,B)操作所否定。往往有时还会出现这种情况。

况，在一个操作执行前必须为真的条件，也会被同一个操作所否定，如 HANDEMPTY 对 pickup 来说就是这样。所以从表中删去这样一些先决条件不会出现问题，于是有：

CLEAR(B): 否定：节点 3 “stack A on B”

断言：节点 4 “Clear B”

根据这张简化表，系统可得出：由于 stack(A, B) 会破坏 stack(B, C) 的先决条件，所以必须先执行 stack(B, C)，因而可得到有定序约束的规划如图 D 所示。

第二个判据是消除冗余先决条件的，可调用来消除子目标的冗余条件。如图 D 中 CLEAR(B) 出现两次有一个是多余的，这样可得到检查修正后的规划如图 E 所示。

下一步进入细节层，要实现 CLEAR(A)，可用 stack(C, x)，把 C 从 A 上拿到别处去，其先决条件为 CLEAR(C)，这样可得图 F 所示的规划。然后再调用消除冲突的判据检查，发现 stack(B, C) 操作将否定 CLEAR(C)，所以 stack(C, x) 必须先执行，可得到图 G 所示的定序约束的规划。再调用消除冗余的判据，检查修正后的规划如图 H。最后系统观察剩余的两个目标 CLEAR(C) 和 CLEAR(B)，这时已与初始状态匹配，所以最后生成的规划如图 I 所示。

这个例子简单说明了分层规划和最少许诺策略相结合的方法，能较直接生成出非线性规划，避免产生太大的搜索树。从过程网的扩展过程看出，最小许诺策略的思想表现在处理子任务相互作用和各种操作之间的定序问题时，在决策条件尚不成熟的情况下，不马上作出决定，并尽量拖延到条件成熟时才作出决策。也就是说，在没有把握能确定两个子任务间操作的顺序时，尽可能保持它们之间的并行性，在过程网扩展到某种程度，能提供排序的充分证据之后，才给出子任务之间操作的定序关系。

还应指出 NOAH 系统也有局限性，有些问题也不能求解，

必须再引入回溯策略才能求解更复杂的问题。

3. MOLGEN系统 (1981)

上面介绍的最小许诺策略在 NOAH 系统中只是用来推迟确定算子的执行顺序，但可以推广应用到一些具有复杂规划过程的系统中所面临的各种决策问题。对于简单的积木世界问题，决策只涉及使用什么算子及其执行的顺序问题。但通常规划除了涉及算子之外，还涉及对象决策问题，即便是积木世界，也会有这种情况，例如，`pickup (A)`，可用`unstack (A, x)` 实现，理论上讲，A 下面的对象可以是任一积木，因此不必急于去作出那一块积木来代替 x 的决策，即可把对象的约束，以变量 x 的形式传播下去，以后再赋值，因为也许还能找到其他实现`pickup (A)` 的更好方法，这就是所谓约束传播 (constraint propagation) 的思想。

对于复杂的领域，关于对象的决策可能成为规划过程的关键问题，MOLGEN 系统就是这种问题规划过程的实例。MOLGEN 是一个设计分子遗传学实验的专家系统，它采用了分层规划方法，其结构有几个特点：

- (1) 以约束条件作为子问题之间相互作用的表示；
- (2) 通过约束条件的传播来发现子问题之间的相互作用；
- (3) 使用显示的问题求解元算子（与领域知识无关）进行具有约束条件的推理；
- (4) 在问题求解中交叉使用最少许诺策略和启发式策略。

一般情况下，层次规划方法是建立在子任务之间具有近似独立的基础上，实际问题子任务之间相互作用不可避免，它们之间的相互约束体现出子任务之间的相互作用，因此相互作用的强弱程度可用其间的约束条件来表示，并可通过约束的传播，逐步对规划进行细化。当某个对象约束较松时，其取值范围较大，约束

传播的结果可不断缩小取值范围，最后达到最少许诺的要求才给予赋值，即此时容易寻到满足所有问题约束的特定值，从而避免了约束变量过早赋值而引起不必要的回溯。可以看出约束传播方法实质上也是最少许诺原则思想的发展。

MOLGEN 系统是从控制结构上把问题求解过程划分为在三个不同层次的空间中进行求解，图 5.11 是 MOLGEN 系统规划过程的空间，每一个空间都有自己的算子及其操作对象。解释程序可以是一个简单的有限自动机，对策略空间的高层算子进行控制，执行这些算子并作用于该层的对象，就生成出元规划（作规划的规划），元规划表示出实验设计方法的一个序列。在设计空间执行元规划算子，就生成出规划，它是元规划的细化，可具体表示出实验步骤的操作序列。在实验空间内执行这个规划，就可按实验步骤完成实验任务，实现了系统目标的要求。

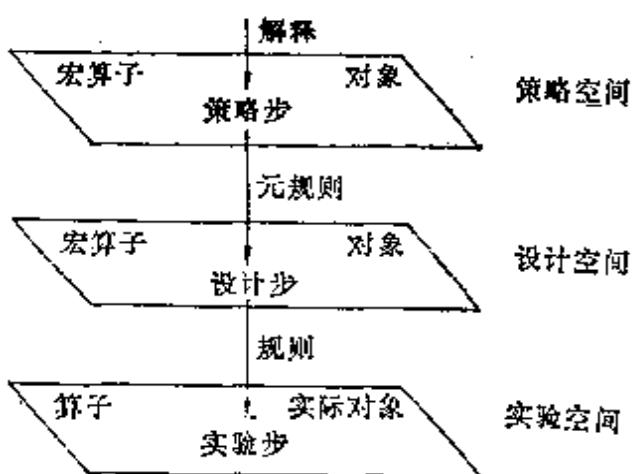


图 5.11 MOLGEN 系统规划过程的空间

5.7 小 结

1. 以上各章讨论了问题求解的几种方法——图搜索法、归约法和归结证明法，这些方法对求解简单问题或复杂问题系统的组成部分都很有用。本章讨论的规划方法也是一种问题求解方法，主要考虑复杂问题分解成子问题后，其间存在相互作用（交连、影响）如何处理以及如何联合各子问题的解来构造出关于原始问题的解等问题。

一般情况下，规划的作用不仅限于执行动作前应事先给出动作执行的次序，而且还可用于监视动作序列的执行过程。

2. 使用目标堆栈进行规划的方法是建立一个目标堆栈来存放子目标及实现它们所提出来的一些算子，然后通过某种控制策略处理子目标相互作用并求得达到目标的动作序列。这种方法生成的规划是由完整子目标规划连接组成，是一种简单的线性规划法，其缺点是有些情况生成的规划不够合理和有效。

3. 使用目标集进行规划的方法是把子目标组成一个集合，使用逆向推理的方式进行求解，求解过程可以灵活选出某个子目标进行处理，这样可以使生成的规划不是由完整子规划的线性序列组成，所以称为非线性规划方法，可以看出生成的规划较合理且有效。

4. 对于很复杂实际人工智能系统，一般要采用多层次规划的思想，在分层规划中要采取各种方法解决子目标互相作用的问题，以降低求解过程的复杂度。在专家系统领域中，NOAH系统和MOLGEN系统就是使用多层次规划的方法建立的专家系统。

习 题

5.1 在 8 数码问题中，设 $\text{right}(x)$ 、 $\text{left}(x)$ 、 $\text{up}(x)$ 、 $\text{down}(x)$ 分别表示 x 单元右边、左边、上面、下面的单元（若这样的单元存在时），试列出 STRIPS 型规则来模拟向上移动 B（空格），向下移动 B，向左移动 B，向右移动 B 等四个动作。

5.2 二曲颈瓶 F_1 和 F_2 的容积分别为 C_1 和 C_2 。合式公式 $\text{CONT}(x, y)$ 表示瓶子 x 包含有 y 容量单位的液体，试写出 STRIPS 规则来模拟以下动作：

- (1) F_1 的所有液体倒到 F_2 中。
- (2) 用 F_1 中的部分液体装满 F_2 。

5.3 用目标堆栈法生成出图 4.19 所示猴子摘香蕉问题的行动规划。

5.4 已知一个 STRIPS 系统的操作集为

skip-exam; P & D; Equal (grade, x)

A: Sleep-late \wedge Equal (grade,

$x = 10$)

write-paper; P; Equal (grade, x)

D; Equal(grade, x) \wedge Sleep-late

A; Equal(grade, x + 50)

系统还有下面两个有关问题域的公理：

GE(grade, 80) \rightarrow ABLE-TO-GRADUATE

LT (grade, 80) \rightarrow NOT(ABLE-TO-GRADUATE)

设该系统试图满足目标

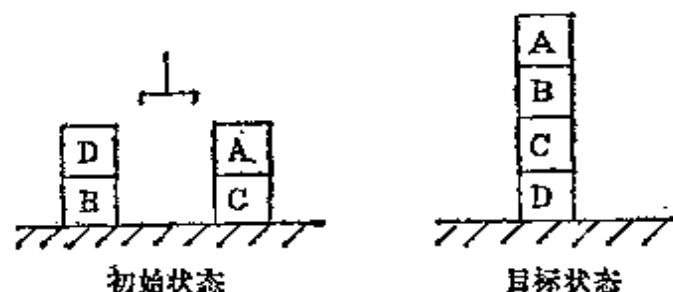
SLEEP-LATE & ABLE-TO-GRADUATE

已知初始条件

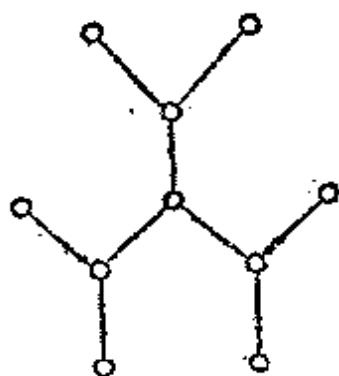
GRADE = 70

试应用目标集方法进行规划，给出搜索示意简图（表示出回溯的作用）和所生成的规划。

5.5 用本章所讨论过的任一规划方法，解决下图所示的机器人摆积木问题，给出所生成的动作规划。



5.6 若在图5.3中将一些节点合并起来，变成为如下结构：



试根据这个结构的形式确定一个分层规划系统，并用一个例子说明系统的工作原理。

5.7 设分层规划系统在某一层求解时失败，关于什么类型的失败原因信息在寻找可供修改的高层规划中可能是有用的？试用一个例子加以说明。

第六章 人工智能语言

在前几章中，讨论了如何运用人工智能技术求解一些问题的一般方法，这些方法可运用于定理证明、自然语言理解、模式识别、机器人规划以及专家系统等各个领域。然而，如何在计算机上实现这些方法呢？这就涉及到了计算机语言问题，也就是说，是否需要一种专门的人工智能语言。严格地讲，可以使用任何计算机语言来实现这些算法，但是大量事实表明，使用专门的人工智能语言，可以方便而有效地建立人工智能系统。

在人工智能的研究发展过程中，从一开始就注意到了语言问题，开发出了适合于各种用途的计算机语言，较早地被作为人工智能语言使用的计算机语言是LISP。LISP语言最初是在1960年作为定义数学函数的一种记法来介绍的，但因其具有较强的符号处理功能和较灵活的控制结构，特别适合于人工智能的研究，从而很快受到了人工智能工作者的青睐，直到今天，LISP语言仍然是应用最为广泛的人工智能语言。

从七十年代开始，在LISP语言的基础上，通过扩充功能的方法，设计出了PLANNER,CONNIVER以及POP-2等语言，同时推出了程序设计环境更为完善的强化LISP版本——INTERLISP。八十年代初，在总结了各种LISP不同版本的优缺点的基础上，又推出了功能更强的COMMONLISP版本。

六十年代末期，以谓词逻辑为基础，开发了运用于定理证明的QA1、QA2和QA3语言，后来受到PLANNER的影响，又推出了QA4语言。

随着功能的扩充，系统变得越来越庞大，这样既降低了处理

速度，又为程序调试带来了困难，为此，于1972年开发了基于一阶谓词逻辑的逻辑型语言——PROLOG，该语言实现了自动推理等功能，是一种比较有前途的计算机语言。

从七十年代中期开始，随着几个具有代表性的专家系统的推出，又产生了各种专门适合于建立专家系统的工具语言，如较典型的EMYCIN、KAS、EXPERT、OPS5以及ART等，将计算机语言推向了一个更高的层次。

本章首先介绍LISP语言，然后介绍PLANNER和PROLOG语言，最后扼要地介绍一下专家系统工具，以便使读者对各种不同的人工智能语言有一个概貌的了解。

6.1 LISP

LISP是一种计算机的表处理语言，它是英文List Processing的缩写。自从1960年J. McCarthy在美国MIT首先发表LISP以来，很快就被人工智能工作者所接受，成为迄今为止，在人工智能领域应用得最广泛的计算机语言。人工智能所取得的各项成果，都是和LISP的功劳分不开的，人们曾这样对LISP语言做出评价：LISP语言是人工智能的数学，不仅对人工智能的机器实现有重要意义，而且是人工智能理论研究的重要工具。LISP语言武装了一代人工智能科学家。

目前，流行的LISP语言具有各种不同的版本，使用最广泛的是INTERLISP、MACLISP和COMMON LISP。以目前的发展看，COMMON LISP将成为一种标准，有统一各种LISP“方言”的趋势。本节将以COMMON LISP为蓝本，对LISP语言做一个简单的介绍。

1. 基本结构

LISP 语言的程序和数据具有统一的结构，即S-表达式。组成S-表达式的基本要素是原子，原子是由一些排列在一起的字符组成的，它含有以下三类：

(1) 文字原子。又名符号 (Symbol)，一般指由英文字母开头的一串字符。如 ABC、append、X12等。

(2) 串原子。串原子是用双引号括起来的一串任意字符。如“*This is an atom*”、“ABC”等。

(3) 数字原子。定点数和浮点数统称为数字原子。如123、3.14等。

S-表达式可以由下面的递归定义得到：

- (1) 原子是S-表达式；
- (2) 两个S-表达式组成的点对，即形式 (<S-表达式1> · <S-表达式2>) 是S-表达式。

由以上的递归定义可知，以下的都是S-表达式。

(A · B) (A · (B · C)) ((A · B) · (C · D))

在LISP语言中，最常用的结构是表，表是一种特殊的S-表达式，其定义如下：

表是由用圆括号包围的若干个元素组成的，各元素之间以分隔符分开，元素可以是原子或者是表。

例如，(A B C)、(x y (A B))、(1 (2 (3)))等都是表。

特别，当表的元素数为零时，我们称之为**空表**。记为 () 或 NIL。

表面上看来，表与S-表达式有很大的不同，实际上表是一种特殊的S-表达式的简写形式，每个表都对应着一个S-表达式。

下面举例说明其对应关系：

(A) \Leftrightarrow (A · NIL)

$(A \ B) \Leftrightarrow (A \cdot (B \cdot NIL))$

$((A \ B) \ C \ D) \Leftrightarrow ((A \cdot (B \cdot NIL)) \cdot (C \cdot (D \cdot NIL)))$

2. 基本函数

LISP 语言不同于 PASCAL、FORTRAN 等过程型语言，它是一种函数型语言，一切功能由函数实现，一个 LISP 程序就是一些函数的集合。

所有 LISP 函数都以表的形式出现，并使用前缀表示方式，每个函数都有一个回送值。

(1) 算术函数

算术运算虽不是 LISP 语言的特长，但由于它比较简单，大家又比较熟悉，因此我们首先从介绍算术函数开始。

算术函数的使用比较简单，只须将要计算的算术表达式转化成相应的前缀形式，用表表示出来就可以了。例如要计算 $3 + 5$ ，只要表示为

$(+ \ 3 \ 5)$

就可以了。

相应地， $(* \ 3 \ 5)$ 计算的是 3×5 ， $(/ \ 3 \ 5)$ 计算的是 $3/5$ 等。由于 LISP 的函数允许嵌套使用，因而可以计算更为复杂的算术表达式。例如

$(+ \ (* \ 3 \ 5) \ (/ \ 4 \ 2)) \Rightarrow 17$

其中“ \Rightarrow ”是为了书写的方便而加上的，它表示“ \Rightarrow ”后面的数字是前面函数的返回值。

(2) 表处理函数

LISP 语言中提供的大部分函数都是与表处理有关的，这里首先介绍一些最常用到的表处理函数。

函数 CAR 的功能是将表的第一个元素作为它的返回值。例如

(CAR '(a b c)) \Rightarrow a
(CAR '((a b) c)) \Rightarrow (a b)

式中符号'表示禁止求值之意，后面还要加以说明。

第一个例子中，a是表(a b c)的第一个元素，因此CAR的返回值是a；第二个例子中，表((a b) c)的第一个元素是表(a b)，因而求CAR的结果是得到(a b)。

CDR是和CAR相对应的函数，它的返回值是一张表，该表中包含原来表中除第一个元素以外的所有元素。例如

(CDR '(a b c)) \Rightarrow (b c)
(CDR '((a b) c)) \Rightarrow (c)

同算术函数一样，表处理函数也可以嵌套使用。例如，求表的第二个元素可以这样进行：

(CAR (CDR '(a b c))) \Rightarrow b

由于经常用到类似于这样的CAR、CDR的组合，为了方便，LISP中提供了形如C \times \times R、C \times \times \times R和C \times \times \times \times R这样的缩写形式，其中每一个 \times ，要么是A——表示CAR，要么是D——表示CDR。这样，上例可以简写为

(CADR '(a b c)) \Rightarrow b

函数CONS有两个参数，当第二个参数是原子时，CONS的功能是将两个参数形成一个点对。

(CONS 'a 'b) \Rightarrow (a . b)

当CONS的第二个参数是一个表时，CONS的功能是将第一个参数作为一个元素，插入到第二个参数中。

(CONS 'a '(b c)) \Rightarrow (a b c)
(CONS '(a b) '(c d)) \Rightarrow ((a b) c d)

函数LIST的参数可以有任意多个，其功能是将这些参数作为表的元素组成一个表。

(LIST 'a 'b 'c) \Rightarrow (a b c)

(LIST (CAR '(X Y)) (CDR '(a b))) \Rightarrow (X (b))

函数 APPEND 可以将两个表联结成为一个表。新表的元素由原来两个表的所有元素组成。

(APPEND '(a b) '(c d)) \Rightarrow (a b c d)

CONS、LIST 和 APPEND 是三个最常用到的构造表的函数，由于其功能相近，极易混淆，请注意其区别。

(3) 求值与赋值

通过前面的介绍我们可以看到，LISP 通过对以 CAR 或 CDR 开头的表进行求值，可以分离一个表结构，对以 CONS 等开头的表求值，达到构造表的目的，而对以 +、-、* 和 / 开头的表求值，则可以进行算术运算等。事实上，LISP 总是试图对一切 S- 表达式进行求值，不仅对表这样，对原子也是如此。不过，表的值是通过函数运算得到的，而原子的值是事先通过赋值函数赋给它的。

一个最基本的赋值函数是 SET，它有两个参数，其功能是使得第二个参数成为第一个参数的值。例如：

(SET 'A '(X Y))

这样使得原子 A 具有值 (X Y)，而且可以在以后的调用中使用它的值。例如

(CAR A) \Rightarrow X

在这里，首先对 A 求值，得到表 (X Y)，然后将 CAR 作用于 (X Y) 得到返回值 X。

已经提到过，LISP 总是试图对每一个 S- 表达式求值，但有时我们想要说明某些部分是原始数据，不需要对它们进行求值，前面已多次使用到的'号所起的就是这个作用。'号告诉 LISP，它后面的一个 S- 表达式是原始数据，不要对它们进行求值。请看下面两个例子，这里假定 x 的值是 (a b c)。

(CAR '(CDR x)) \Rightarrow CDR

(CAR (CDR x)) \Rightarrow b

第一个例子中，由于'号的存在，(CDR x)被认为是原始数据，不进行求值，CDR是它的第一个元素，因此CAR作用的结果得到CDR；第二个例子中，没有'号，首先对x求值得到(a b c)，然后CDR作用于该表得到(b c)，最后CAR作用得到b。

再举几个例子，以加深对求值与不求值的理解。

(SET 'l '(a b c)) ; l得到值(a b c)

(SET 'n (CDR l)) ; n得到值(b c)

(SET (CAR l) 'x) ; a得到值x

(APPEND l n) \Rightarrow (a b c b c)

(LIST 'l 'n) \Rightarrow (l n)

(LIST l n) \Rightarrow ((a b c)(b c))

正象这里看到的一样，我们经常遇到要为一个原子赋值，为了使用起来更方便，LISP中提供了另一个赋值函数SETQ，它同SET的功能一样，只是SETQ能自动地不对第一个参数求值。

(SETQ A '(x y))

这样A将得到值(x y)，它同

(SET 'A '(x y))

等价。

COMMON LISP中提供的另一个赋值函数是SETF，它具有很强的功能，除能代替SETQ使用外，还能为某个具体的位置赋值。

(SETF x '(a b)) ; x赋值为(a b)

(SETF (CAR x) 'c) ; x的CAR部分被c代替了，也就是说x变为(c b)了。

(4) 谓词函数

所谓谓词函数就是其回送值取真或假的函数，在LISP中真

用T表示，假用NIL表示，当函数的回送值为非NIL时，也表示为真。T和NIL是两个特殊的原子，(NIL比较特殊，它既是一个空表，又是一个原子)，它们的值分别为其自己。

ATOM函数是判断其参数是否为原子的谓词，当参数为原子时，返回值为T，否则为NIL。

(ATOM ' a) \Rightarrow T

(ATOM '(a b)) \Rightarrow NIL

(ATOM (CAR '(a b))) \Rightarrow T

(SETQ L '(x y))

(ATOM L) \Rightarrow NIL

(ATOM ' L) \Rightarrow T

LISTP函数在其参数为表时取T，否则取NIL

(LISTP '(a b)) \Rightarrow T

(LISTP NIL) \Rightarrow T

(LISTP 'a) \Rightarrow NIL

(SETQ L '(x y))

(LISTP L) \Rightarrow T

(LISTP (CDR L)) \Rightarrow T

NULL函数则用于判断其参数是否为空表。

(NULL NIL) \Rightarrow T

(NULL '(a b)) \Rightarrow NIL

(NULL (CDR '(a))) \Rightarrow T

用于比较两个S-表达式是否相等的谓词有三个，它们既有相同之处，又各有区别，请注意它们的不同用法。

EQ用来比较两个S-表达式是否相等，但其比较方法是通过判断指向两个S-表达式的指针是否相等来进行的，只有当指针相同时，EQ才为真。因此，当两个S-表达式逻辑上相等时，不一定EQ相等。在LISP中，文字原子的存储是唯一的，因此对于

文字原子，其逻辑相等与 EQ 相等是等价的。

```
(EQ 'a 'a) => T
(EQ 'a (CAR '(a b))) => T
(EQ '(a b) '(a b)) => NIL
```

另一个相等谓词是 EQL，对 EQ 相等的也一定是 EQL 相等。EQL 比 EQ 多了一个功能，它可以判断两个数字原子是否相等。当两个数字原子逻辑上相等，且存储结构相同时，则 EQL 相等。

```
(EQL 'a (CAR '(a b))) => T
(EQL '(a b) '(a b)) => NIL
(EQL 3 3) => T
(EQL 3.5 3.5) => T
(EQL 3 3.0) => NIL
```

最通用的相等谓词是 EQUAL，只要它的两个参数在逻辑上相等，则 EQUAL 相等。

```
(EQUAL 'a 'a) => T
(EQUAL '(a b) '(a b)) => T
(EQUAL '(a b) (CONS 'a '(b))) => T
```

MEMBER 函数测试它的第一个参数是否是第二个参数（通常是一个表）的元素。当第一个参数是第二个参数的元素时，则 MEMBER 的返回值为从第二个参数中与第一个参数相同的那个元素开始的余下的部分，否则返回值为 NIL。这里进行测试的默认函数为 EQL。

```
(MEMBER 'b '(a b c)) => (b c)
(MEMBER 'x '(a b c)) => NIL
(MEMBER '(a b) '((1 2) (a b) (c d))) => NIL
```

第三个例子之所以返回值为 NIL，是由于默认的测试函数是 EQL 的原因。若希望用其它的函数进行测试，如使用 EQUAL，

则需要在调用 MEMBER 时，加上测试说明。

```
(MEMBER '(a b) '((1 2) (a b) (c d))) : test  
'EQUAL) => ((a b) (c d))
```

函数 NUMBERP 测试其参数是否为一个数字。

```
(NUMBERP 3) => T  
(NUMBERP 'a) => NIL  
(SETQ one 1)  
(NUMBERP one) => T
```

函数 > 和 < 可以用来比较两个数的大小。

```
(> 3 5) => NIL  
(> 4 2) => T  
(< 3 5) => T  
(< 4 2) => NIL
```

ZEROP 函数则要测试其参数是否为 0。

```
(ZEROP 0) => T  
(ZEROP 1) => NIL
```

函数 AND 和 OR 被用作逻辑运算，它们均可以有任意多个参数。只有当所有参数均为非 NIL 时，AND 返回最后一个参数的非 NIL 值，否则 AND 的值为 NIL。只有当所有参数均为 NIL 时，OR 返回 NIL 值，否则 OR 的值为第一个非 NIL 参数值。

```
(AND T T NIL) => NIL  
(AND (CAR '(a b)) (CDR '(x y))) => (y)  
(OR T T NIL) => T  
(OR (MEMBER 'a '(x y)) (CDR '(x))) => NIL
```

(5) 条件函数

根据不同的条件，执行不同的操作，这就是条件函数的功能。在 LISP 中，最常用到的条件函数是COND。COND 的一般调用形式如下：

```
(COND (<测试1>…<结果1>)
      (<测试2>…<结果2>)
      :
      (<测试i>…<结果i>)
      :
      (<测试n>…<结果n>))
```

COND 函数含有 n 个测试条件和 n 个可能返回的结果。在调用执行时，COND 首先从上到下依次对测试条件进行求值，直到找到一个非 NIL 的测试值，假定是 <测试i>，然后依次对 <测试i> 后面的 LISP 式子进行求值，并将最后一个 LISP 式子——<结果i> 的返回值做为 COND 函数的返回值。若没有一个测试的值为非 NIL，则 COND 的返回值为 NIL。

```
(COND ((NULL x) 0)
      ((ATOM x) 1)
      ((LISTP x) (LENGTH x)))
```

在该例中，若 x 的值为 NIL，则 COND 的返回值为 0；若 x 为原子，则 COND 的返回值为 1；若 x 的值为 表，则 COND 的返回值为 表的长度。

3. 函数的定义

我们已经介绍了许多 LISP 的基本函数，这些函数虽然包含了许多功能，但只能进行一些规定操作，不能解决更多的问题。为此，LISP 提供了 DEFUN 这个函数，允许用户自己定义所需要的函数，来解决用户自己的问题。

DEFUN 的调用格式如下：

```
(DEFUN<函数名> (<形参表>)
  {函数定义体})
```

例如我们想定义一个函数，它把表中没有的新元素加到表的前面。如果这个新元素已在原来的表中，就直接返回原来这个表。

完成该功能的函数，我们把它起名为 AUGMENT。其定义如下：

```
(DEFUN AUGMENT (item bag)
  (COND((MEMBER item bag) bag)
        ;已在原表中，返回原表。
        (T (CONS item bag)))))
        ;把item加到表的前面。
```

当这样定义好之后，我们可以象调用那些基本函数一样使用 AUGMENT 函数：

```
(AUGMENT 'a '(b c))=>(a b c)
```

(1) 函数的递归定义

为了定义出更多、功能更强的函数，需要一些手段。LISP 中最常用到的函数定义手段是递归。

所谓递归，就是函数自己调用自己。这就表示，至少有一个最简情况是可以解决的，另外，每递归一步都要使问题化简一步，并且化简的最终结果可以达到最简情况，否则将无穷递归下去。

下面我们通过例子来说明如何使用递归。最简单的例子就是数学函数本身的定义就是递归的。

例如，阶乘函数的数学定义为

$$\begin{aligned}0! &= 1 \\1! &= 1 \\n! &= (n-1)!n\end{aligned}$$

根据此定义，我们可以很容易地写出其 LISP 定义。

```
(DEFUN N! (n)
  (COND ((=n 0) 1); n等于0时为1
        ((=n 1) 1); n等于1时为1
        (T (*n (N! (- n 1))))); 递归
```

当对(N! 4)进行求值时，其递归过程可以用图6.1来表示。图中的每一个○表示函数的一次调用，○中的数字表示调用的先后

次序。向下的箭头进入圆圈，表示调用函数，该箭头左方的数字为这次调用的函数的参量值。从圆圈向上发出的箭头表示函数此次调用的返回值，其值标在箭头的右方。

数学上另一个递归定义的函数的例子是 FIBONACCI 函数，其定义如下：

$$f(0) = 1$$

$$f(1) = 1$$

$$f(n) = f(n-1) + f(n-2) \quad n > 1$$

用 LISP 表示就是：

```
(DEFUN FIBONACCI (n)
  (COND ((= n 0) 1)
        ((= n 1) 1)
        (T (+ (FIBONACCI (- n 1))
               (FIBONACCI (- n 2))))))
```

图 6.2 给出的是当 $n=4$ 时的递归过程图。

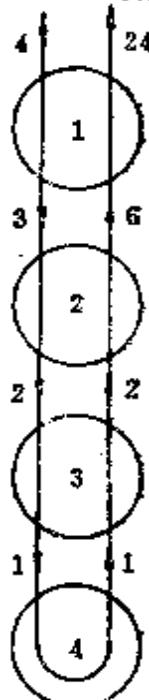


图 6.1 $(N_1 4)$ 的递归过程

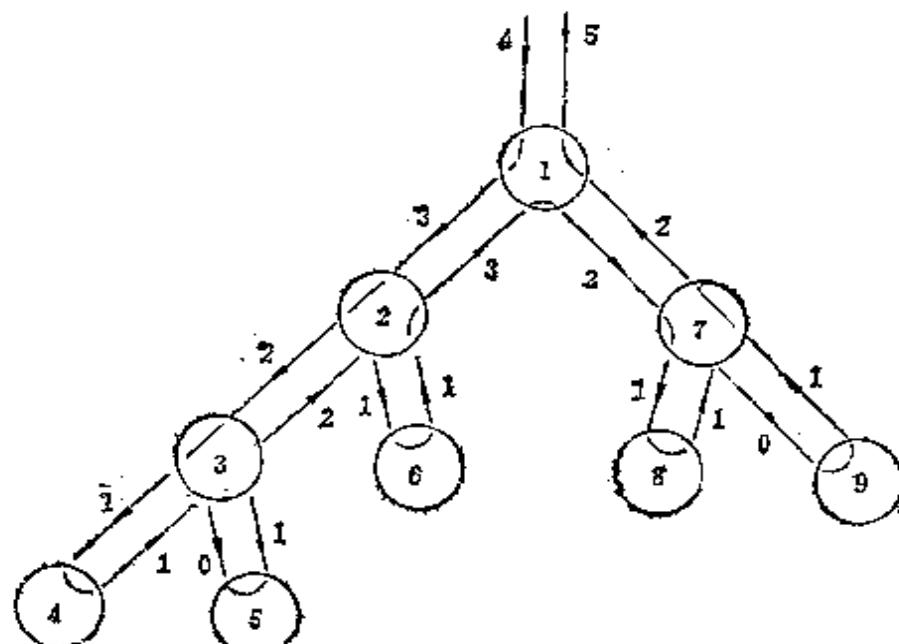


图 6.2 $(FIBONACCI 4)$ 的递归过程

在看过递归在两个简单的数学问题上的应用之后，我们对递归的用法有了一个大致的了解。下面我们来看一看，如何使用递归来解决一些更复杂的问题。

首先，我们假定 LISP 中没有 MEMBER 这个函数，我们通过定义得到它。

```
(DEFUN MEMBER (item s)
  (COND ((NULL S) NIL)
        ((EQUAL item (CAR S)) S)
        (T (MEMBER item (CDR S)))))
```

为了递归能够结束，MEMBER 首先考虑两种最简情况：①当 S 为空时，则 item 一定不在 S 中出现，因此 MEMBER 的回送值为 NIL；②当 S 的第一个元素等于 item 时，MEMBER 成功，并以 S 为其回送值。当这两种最简情况都不成立时，MEMBER 进行递归调用，虽然这时问题并没有得到求解，但被简化了一步，所要处理的表比原来少了一个元素。

下面再举一个对表中的所有原子进行计数的例子。注意这里是对表内的所有原子计数而不是元素，也就是说，当表的某个元素是表时，还需要深入到该元素的内层进行计数。

```
(DEFUN COUNTATOMS (S)
  (COND ((NULL S) 0)
        ((ATOM S) 1)
        (T (+ (COUNTATOMS (CAR S))
              (COUNTATOMS (CDR S))))))
```

首先，COUNTATOMS 定义最简情况即 NIL 的原子个数为 0，原子的原子个数为 1，其它情况则递归调用，表头的原子个数加上表尾的原子个数即为整个表的原子个数。

(2) 函数的迭代定义

迭代也是一种主要的函数定义手段，尤其是熟悉象 PASCAL

这样的过程型语言的用户，可能更习惯于使用迭代而不是递归。使用迭代往往比使用递归效率高和节省内存，但有些问题使用递归要比使用迭代简单、明了。如上面定义过的 COUNTATOMS 函数，若只单纯地使用迭代，其定义要复杂得多。而且递归是“纯”的 LISP 定义手法，迭代只是为了增加更多的定义手段才增加到 LISP 中来的。

最常用到的迭代，是通过 PROG 函数实现的，PROG 函数本身没有什么具体的含义，它只起到一种可以进行迭代的媒介作用。在 PROG 函数内，可以使用 GO 函数转到某个给定的标号，也可以通过 RETURN 函数退出 PROG，并使得 PROG 的返回值为 RETURN 的参数值。

例如，使用 PROG 迭代定义阶乘函数：

```
(DEFUN N! (n)
  (PROG ((result 1))
    LOOP
      (COND ((= n 0) (RETURN result))
            ((= n 1) (RETURN result))
            (T (SETQ result (* n result))
                (SETQ n (- n 1))
                (GO LOOP)))))
```

紧跟在 PROG 后面的表，说明了 result 是一个局部变量，其初始值为 1，下面的 LOOP 是一个标号，当 $n=0$ 或 $n=1$ 时，通过 RETURN 函数，回送 result 的值，其它情况，将 n 乘到 result 上去，并得到 n 减 1，使用 GO 函数返回到标号 LOOP，如此循环往复，直到结束为止。

LISP 中还提供了一类隐式迭代函数，这一类函数的函数名均以 MAP 开头，故称它们为 MAP 类函数。MAP 类函数的一个典型代表是 MAPCAR 函数。

MAPCAR 函数的第一个参量是一个要调用的函数名，其它参量均为表，其它参量的个数等于第一个参量（它是一个函数名）所需要的参量数。MAPCAR 的功能是，从第二个参量开始，依次取出各个参量的第 i 个元素 ($1 \leq i \leq n$, n 是各个参量中最短的那个表的长度)，然后把它们作为第一个参量的参量进行求值，每次求值的结果形成一张表作为 MAPCAR 的回送值。

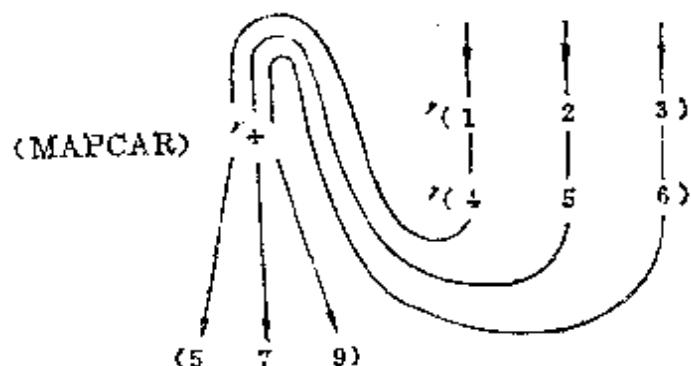


图 6.3 MAPCAR 的作用方式

$(MAPCAR '+ '(1 2 3) '(4 5 6)) \Rightarrow (5 7 9)$

它的作用方式可用图6.3表示。

下面，我们使用 MAPCAR 函数，给出前面已经定义过的 COUNTATOMS 函数的更简洁的定义。

```
(DEFUN COUNTATOMS (s)
  (COND ((ATOM s) 1)
        (T (APPLY '+ (MAPCAR 'COUNTATOMS
                               s))))))
```

这里我们用到了 APPLY 函数，通常 APPLY 有两个参量，第一个参量是一个函数名，第二个参量是一张表，APPLY 的功能是将表中的所有元素做为函数的参量进行求值，并将函数的返回值作为其自己的返回值。

```
(SETQ L '(1 2))
(APPLY '+ L) \Rightarrow 3
```

有时为了完成更复杂一些的操作，在 MAPCAR 函数中经常要用到 LAMBDA 式子。LAMBDA 式子可以定义匿名函数，凡

是在要求用函数名作为参量的地方，均可以用 LAMBDA 式子代替。

LAMBDA 式子的格式如下：

(LAMBDA (<形参表>) {<函数定义体>})

作为使用 LAMBDA 的例子，我们把 COUNTATOMS 重新定义如下：

```
(DEFUN COUNTATOMS (s)
  (APPLY '+ (MAPCAR '(LAMBDA (x)
    (COND ((ATOM x) 1)
          (T (COUNTATOMS x)))))
    s)))
```

该定义的思路是：将 s 中的各个元素交给 MAPCAR 去处理，若某个元素是原子，就记数1，若不是原子（一定是表）就递归调用 COUNTATOMS 来处理，然后将计算结果加起来作为 COUNTATOMS 的返回值。

4. 宏的定义

宏在 LISP 中起着举足轻重的作用，LISP 中的许多功能，正是因为有了宏，才得以实现。定义宏的格式与定义函数的格式完全相同，只是将 DEFUN 换成了 DEFMACRO，它们的差别在于以下两点：

① 函数的形参值等于其相应的实参的值，而宏的形参值则等于其相应的实参本身。这就是说，函数将对其调用参数求值，而宏则不对其调用参数求值。

② 函数对其定义体求值，并直接将求值结果作为该函数的回送值，而宏则对其定义体的求值结果再进行一次求值，并把第二次求值结果作为该宏的回送值。通俗一点说就是，函数对其定义体进行一次求值，而宏对其定义体进行两次求值。

那么，为什么要引入宏呢？下面我们通过定义 IF，来说明这个问题。IF 有三个参数，我们所希望的功能是：当第一个参数为真时，执行第二个参数，否则执行第三个参数。

看到这个要求之后，很自然地想到要这样来定义 IF：

```
(DEFUN IF (test then else)
  (COND (test then)
        (T else)))
```

然而这个定义并不能完成我们所需要的功能，其原因就是函数首先对其参数进行求值。下面通过一个具体的调用例子来说明为什么这个定义不能满足要求。

```
(IF (ATOM X) (SETQ Y 1) (SETQ Y 2))
```

我们本来希望当 X 为原子时 Y 赋值 1，否则的话 Y 赋值 2。但由于 IF 已被定义为函数了，因此首先要对所有参数求值，其结果导致了无论 X 是原子与否，(SETQ Y 2) 都将被求值，使得 Y 永远被赋值为 2。

而用宏则可以解决这个问题，下面给出 IF 的宏定义。

```
(DEFMACRO IF (test then else)
  (CONS 'COND (list (list test then)
                     (list T else))))
```

还是以 (IF (ATOM x) (SETQ Y 1) (SETQ Y 2)) 为例，来说明宏 IF 是如何正确地工作的。

首先宏不对其参数求值，并且形参的值就是实参本身。这样，形参 test 的值为 (ATOM X)，then 的值为 (SETQ Y 1)，else 的值为 (SETQ Y 2)。因而，对 IF 的定义体的第一次求值结果为：

```
(COND ((ATOM X) (SETQ Y 1))
      (T (SETQ Y 2)))
```

由于 IF 是宏，还要对第一次的求值结果进行第二次求值，也就

是对上述的 COND 函数求值，显然当第二次求值时，IF 达到了我们所希望的结果。

虽然上述 IF 的宏定义能满足要求，但其定义比较复杂，可读性差。为此，在COMMON LISP 中提供了“`”、“,”和“,@”三个符号，这三个符号，为简化宏定义，提高可读性，起到了关键的作用。

符号“`”同“,”一样具有阻止求值的功能，但在“`”的管辖范围内，当遇到符号“,”或“,@”时，要对“,”或“,@”后面的S-表达式求值，并且对于“,@”，当求值结束后，对求值结果要去掉一层括号。

```
(SETQ L '(a b))  
'(x y ,L) => (x y (a b))  
'(x y ,@L) => (x y a b)
```

使用这些功能，我们可以得到 IF 的更简洁的宏定义，显然其可读性被改善了。

```
(DEFMACRO IF (test then else)  
  (COND (,test ,then)  
        (T ,else)))
```

5. 特性

在 LISP 中，文字原子允许有特性值，并且允许每个原子具有多个不同的特性值。特性的使用，为解决许多问题带来了方便。例如，要进行人事管理，就要记录人的年龄、性别等，为此，我们可以用 sex 特性来记录性别，而用 age 特性记录年龄。

假设 Wang 的 age 特性为 60，sex 特性为 male，则可以用 GET 函数分别得到其特性值：

```
(GET 'Wang 'age) => 60  
(GET 'Wang 'sex) => male
```

组合使用 SETF 和 GET，可以为某个原子的特性赋值，如为 LI 的 age 特性赋值 50：

```
(SETF (GET 'LI 'age) 50)
```

最后，作为一个应用特性的例子，我们定义一个判断两个集合是否相等的函数，其中特性的使用降低了算法的复杂性。但在这里，我们假定了集合的元素只是由文字原子组成的。

```
(DEFUN SET-EQUAL (set1 set2)
  (MAPCAR '(LAMBDA (x)
    (SETF (GET x 'set) T)) set1)
  (COND ((AND (EVERY '(LAMBDA (x)
    (GET x 'set)) set2)
    (= (LENGTH set1) (LENGTH set2)))
    (MAPCAR '(LAMBDA (x)
      (SETF (GET x 'set) NIL)) set1)
    T)
    (T (MAPCAR '(LAMBDA (x)
      (SETF (GET x 'set) NIL)) set1)
    NIL))))
```

在定义中，我们首先用 MAPCAR 为 set1 的每个元素赋 set 特性值为 T，然后进行判断，当集合 set1 和 set2 的元素个数相等，并且 set2 的每个元素的 set 特性值均为 T 时，则集合 set1 和 set2 相等，先删除 set1 所有元素的 set 特性值，然后回送值 T，否则也先删除 set1 所有元素的 set 特性值，然后回送值 NIL。其中函数 EVERY 的功能是将它的第一个参数（通常是一个函数名）依次作用到第二个参数（通常是一张表）的每个元素上，当得到的每个结果均为非 NIL 时回送值为 T，否则为 NIL。

6. 程序设计举例

(1) 快速排序

排序是经常遇到的一种操作，它有很多不同的算法，下面我们通过快速排序的例子，来说明如何用 LISP 编写排序程序。

快速排序的思想是这样的：首先以表的第一个元素 x 为基准，将表中比 x 大的元素放入一个表中，把比 x 小的元素放入另一个表中，然后将这两个表分别进行排序，并将排序结果同 x 一起进行合并，作为总的排序结果回送。

```
(DEFUN QUICK-SORT (I)
  (COND ((NULL I) NIL)
        (T (APPEND (QUICK-SORT (LARGE I))
                    (LIST (CAR I))
                    (QUICK-SORT (SMALL I))))))

(DEFUN LARGE (I)
  (MAPCAN '(LAMBDA (x)
    (COND ((> x (CAR I)) (LIST x))
          (T NIL)))
    (CDR I)))

(DEFUN SMALL (I)
  (MAPCAN '(LAMBDA (x)
    (COND ((<= x (CAR I)) (LIST x))
          (T NIL)))
    (CDR I))))
```

函数 QUICK-SORT 是进行快速排序的主函数，LARGE 用来求出 I 中比 I 的第一个元素大的所有元素，SMALL 用来求出 I 中小于等于 I 的第一个元素的所有元素。

其中，MAPCAN 函数是一个系统函数，其功能基本同 MAPCAR，只是在完成 MAPCAR 之后，MAPCAN 对 MAPCAR 的回送值 s 进行一些处理，其处理方法是：若 s 的某个元素是表，

则该表的元素是MAPCAN 的回送表中的元素，若 s 的某个元素是 NIL 或原子，则该元素不在 MAPCAN 的回送表中出现。

试比较以下两个结果：

```
(MAPCAR 'CAR '(((a) b)(x y)(NIL z) ((1 2) 3)))
⇒ ((a) x NIL (1 2))
(MAPCAN 'CAR '(((a) b) (x y)(NIL z)((1 2) 3)))
⇒ (a 1 2)
```

(2) 形式微分

所谓形式微分，就是符号微分，它是和数字微分相对应的，它要求输入是一个数学式子，输出是对该数学式子的微分。为了方便起见，约定数学式子全部用前缀形式表示。若用 (D e x) 来表示 e 对 x 的微分，则函数 D 可以定义如下：

```
(DEFUN D (e x)
  (COND ((ATOM e) (IF (EQ e x) 1 0))
        (T (APPLY (D-RULE (CAR e))
                  (APPEND (CDR e)
                           (LIST x)))))))
```

其中，D-RULE是一个获取给定操作符的微分规则的函数。微分规则的存放，是通过为相应操作符建立 d 特性的方法完成的，因而 D-RULE的定义很简单：

```
(DEFUN D-RULE (operator)
  (GET operator 'd))
```

操作符的d特性值，事先用 SETF 函数建立好。例如，对于 加法和乘法在数学上有

$$\frac{d(u + v)}{dx} = du/dx + dv/dx$$

$$\frac{d(u \cdot v)}{dx} = v \cdot du/dx + u \cdot dv/dx$$

则用 LISP 表示出来就是

```
(SETF (GET '+ 'd) #'(LAMBDA (u v x)
  `(+ ,(D u x) ,(D v x))))
(SETF (GET '* 'd) #'(LAMBDA (u v x)
  `(* ,(D u x) ,v)
  (* ,(D v x) ,u))))
```

有了这些规则之后，我们就可以求形式微分了，例如

```
(D '/(+(* 2 x)(* x x)) 'x)
=>(+(+(* 0 x)(* 1 2)) (+(* 1 x)(* 1 x)))
```

该结果虽然没有得到化简，但显然是正确的。

这个程序的好处是，在增加新的操作符时，不需对程序本身进行修改，只需要对新的操作符添加上相应的微分规则就可以了。下面通过增加“/”（除法）操作符的例子，说明如何添加微分规则。

首先写出“/”的数学上的微分法则：

$$\frac{d(u/v)}{dx} = (v \cdot du - u \cdot dv) / v^2$$

第二步，用涉及到的参量 u、v、x 作为形参，用 LAMBDA 式子表示出等号右边的内容，其中 ds 用 (D s x) 代替：

```
(LAMBDA (u v x)
  (/ (- (* v (D u x)) (* u (D v x)))
    (* v v)))
```

然后在 LAMBDA 式的定义体的最外层加符号“`”，在 u、v、(D u x) 和 (D v x) 前加“，”号（在 (D u x)、(D v x) 内的 u、v 前不加“，”号）：

```
(LAMBDA (u v x)
  `(/ (-(* ,v ,(D u x)) (* ,u ,(D v x)))
    (* ,v ,v)))
```

最后，将上述 LAMBDA 式子赋给/的 d 特性：

```
(SETF (GET '/ 'd) '(LAMBDA (u v x)
  '(/ (~ (* ,v ,(D u x))
        (* ,u ,(D v x)))
        (* , v, v))))
```

这样就完成了一条新规则的加入。

（3）梵塔问题

我们用递归算法解决梵塔问题，其算法很简单：假定要把 n 个盘子从 A 搬到 C，则首先将 A 上的前 (n-1) 个盘子搬到 B 上，然后将 A 上的最后一个盘子搬到 C 上，最后将 B 盘上的 (n-1) 个盘子搬到 C 上。对于如何实现 (n-1) 个盘子的搬动问题，则借助于递归来实现，直到 n 等于 1 时才实现真正的搬动。可以证明，这样的搬动算法是可以满足要求的。

```
(DEFUN HANOI (a b c n)
  (COND ((= n 1) (MOVE-DISK a c))
        (T (HANOI a c b (- n 1))
            (MOVE-DISK a c)
            (HANOI b a c (- n 1)))))

(DEFUN MOVE-DISK (from to)
  (TERPRI)
  (PRINC "Move Disk From")
  (PRINC from)
  (PRINC "To")
  (PRINC to))
```

为了显示盘子的搬动过程，利用 MOVE-DISK 函数将搬动盘子的次序显示出来，其中 TERPRI 是回车换行函数，PRINC 是显示参数的函数。下面是 n=3 时的求解结果：

```
(HANOI 'a 'b 'c 3)
```

```
MOVE DISK FROM A TO C
MOVE DISK FROM A TO B
MOVE DISK FROM C TO B
MOVE DISK FROM A TO C
MOVE DISK FROM B TO A
MOVE DISK FROM B TO C
MOVE DISK FROM A TO C
```

(4) 传教士与野人问题

对于 $n=3$ 、 $k=2$ 的传教士与野人问题，我们用一个只表示左岸状态的三元组来表示：

$(M\ C\ B)$ $M, C \in \{1, 2, 3\}$, $B \in \{0, 1\}$

其含义是有 M 个传教士， C 个野人， B 条船在左岸。其过河规则也用三元组表示：

$(M_i\ C_i\ 1)$ $M_i, C_i \in \{0, 1, 2\}$ 且 $0 < M_i + C_i \leq 2$

其含义是有 M_i 个传教士， C_i 个野人坐一条船过了河（从左岸到右岸或从右岸到左岸）。所有规则只有五条，我们把它们形成一张表赋给 rule-set：

$(SETQ rule-set'((2\ 0\ 1)(1\ 1\ 1)(1\ 0\ 1)(0\ 1\ 1)(0\ 2\ 1)))$

下面我们用深度优先法求解该问题。

首先定义出判断一个状态是否安全的函数 SAFE，经分析，不难得出只有以下三种状态是安全的：

- ① 左岸的传教士与野人人数相等；
- ② 所有传教士都在左岸；
- ③ 所有传教士都在右岸。

因此得到 SAFE 的定义如下：

```
(DEFUN SAFE (state)
  (COND ((AND (>= (CAR state) 0)
                (<= (CAR state) 3)
```

```

(>= (CADR state) 0)
(<= (CADR state) 3)
(OR (= (CAR state) (CADR state))
    (= (CAR state) 3)
    (= (CAR state) 0))) T)
(T NIL)))

```

当选一条规则之后，从当前状态到下一个状态的转换由 MOVE 函数来完成：

```

(DEFUN MOVE (rule)
  (COND((=(CADDR state) 1) ; 船在左岸时
        (MAPCAR' ~ state rule))
        (T(MAPCAR' + state rule)))) ; 船在右岸时

```

我们设置了一个堆栈 path 来记录解的路径，最初它只有初始状态一个元素，然后每扩展出一个新的合法状态，都将被压入到 path 中。我们设置的另一个堆栈是 rule-set，它的每一个元素都是一张表，rule-set 的第 i 个元素记录的是 path 中第 i 个状态到目前为止还没有使用过的那些规则，以便于回溯时扩展其它的分支。

我们用 FIND-RULE 函数来找到可施加于当前状态的下一条规则，其输入参数是对当前状态所有可使用的规则形成的表 rules，当 rules 空时，说明当前状态已没有可使用的规则，需要进行回溯，这就要改变堆栈 path 和 rule-set 及当前状态 state 的值；当 rules 不空时，则取出 rules 中的第一条规则作为所找的规则。因这条规则将被施加于当前状态，因此应从当前状态可使用的规则集中删除此规则。下面就是 FIND-RULE 函数的定义：

```

(DEFUN FIND-RULE (rules)
  (COND ((NULL rules) ; 当前状态没有可用规则时
        (POP path) ; 回溯

```

```
(SETQ state (CAR path)) ; 设置新的当前状态
(FIND-RULE (POP rules-set))) ; 递归，为新的
                                当前状态查找可用规则
(T (PUSH (CDR rules) rule-set)) ; 去掉 rules 的
                                第一条规则，压入 rule-set 中
(CAR rules))) ; 以 rules 的第一条规则作为找到
                                的规则
```

其中，函数 POP 的功能是进行退栈操作， PUSH 的功能是进行压栈操作。

定义了以上这些函数之后，就可以给出求解传教士与野人问题的主函数 M-C 的定义了。其算法是这样的：如果当前状态就是目标状态，则结束，并输出解路径；否则，查找出一条规则，并用该规则得到一个新状态，若该状态不是一个安全状态，或者是一个已经出现过的状态，就什么操作也不进行，重复以上操作；否则的话，设置该状态为当前状态，并把它记录在路径中，然后把它所有可以使用的规则压到 rules-set 堆栈中。因该状态是一个刚刚产生的状态，则所有的规则它都可以使用。重复以上操作，直到结束。下面就是 M-C 的定义：

```
(DEFUN M-C (state)
  (LET((rules-set (LIST rule-set))
       (path (LIST state)))
    (LOOP
      (COND ((EQUAL state '(0 0 0))
             ; 如果找到目标状态
             (RETURN (REVERSE Path)))
            ; 则成功返回
            (T (LET ((x (MOVE (FIND-
                                RULE
```

```
(POP rules-set)))))  
; 否则, 运用找到的规  
则, 得到一个新状  
态, 并赋给 x.  
(COND ((AND (SAFE  
x); 若 x 是一个安全状态  
(NOT (MEMBER x  
path : TEST  
/EQUAL)))  
; 并且 x 不在 path 中  
(SETF state x); 则以  
x 为当前状态  
(PUSH x path)  
(PUSH rule-set rules  
-set))))))))  
; 设置 x 的所有可用规则
```

其中, 函数 LET 的调用格式为

```
(LET ((x1 y1) … (xn yn))  
<LISP式子1>  
:  
<LISP式子m>)
```

其功能是定义局部变量 x_i ($i=1, \dots, n$), 并使得 x_i 的初始值为 y_i 的值。然后对后面的 m 个 ($m > 0$) LISP 式子分别进行求值, 并以最后一个 LISP 式子的求值结果为其回送值。

函数 LOOP 的调用格式如下:

```
(LOOP <LISP式子1> … <LISP式子n>)
```

其功能是反复地依次对 $<LISP式子1> \dots <LISP式子n>$ 进行求值, 直到遇到 RETURN 函数被执行为止, 并以 RETURN 的

回送值为 LOOP 自己的回送值。

REVERS 的功能是将一个表的所有元素的排列 位 置颠倒一下，即原来的第一个元素成为最后一个元素，原来的最后一个元素成为第一个元素。

应用 M-C 函数求解传教士与野人问题的结果如下：

```
(M-C '(3 3 1))=>((3 3 1) (2 2 0) (3 2 1)
                      (3 0 0) (3 1 1) (1 1 0)
                      (2 2 1) (0 2 0) (0 3 1)
                      (0 1 0) (1 1 1) (0 0 0))
```

6.2 PLANNER

PLANNER 是建立在 LISP 的基础之上，为问题求解及定理证明而提出的一门语言，该语言能够实现传统的正向推理和逆向推理等功能，其思想是在 1968 年由 MIT 的学生 C.Hewitt 提出的，但完整的 PLANNER 始终没有实现，现有的只是它的一个子集 MICRO-PLANNER。MICRO-PLANNER 虽然只包含了 PLANNER 的一些基本功能，但却曾在许多问题求解程序中被当作程序的基础而使用。

一个 PLANNER 程序包括断言和定理两部分，它们分别被存储在断言数据库和定理数据库中。下面举例说明如何表示一个断言。

假定在桌面上有长方体 BLOCK1、BLOCK2 和四面体 PYRAMID1、PYRAMID2。用 (BLOCK BLOCK1) 来表示 BLOCK1 是长方体，为将该断言加入到断言数据库中，则需要进行如下操作：

```
(THASSERT (BLOCK BLOCK1))
```

同样可以将其它几个物体加入到断言数据库中：

```
(THASSERT (BLOCK BLOCK2))  
(THASSERT (PYRAMID PYRAMID1))  
(THASSERT (PYRAMID PYRAMID2))
```

其中 THASSERT 是一个将断言加入到断言数据库中的函数，为了便于区分，在 PLANNER 的函数名前均有 TH。

另外，若假定 BLOCK1、PYRAMID1 的颜色为红色，而 BLOCK2、PYRAMID2 的颜色为蓝色，则同样可将它们加入到断言数据库中：

```
(THASSERT (COLOR BLOCK1 RED))  
(THASSERT (COLOR BLOCK2 BLUE))  
(THASSERT (COLOR PYRAMID1 RED))  
(THASSERT (COLOR PYRAMID2 BLUE))
```

PLANNER 的定理分为三种，运用定理可以描述出如何根据已知事件推导出新的结论。

（1）后件定理

后件定理实际上描述的是一种逆向推理。它首先给出一个结论，然后给出该结论所要满足的前提。例如

```
(THCONSE (x) (FLAT ? x))  
          (THGOAL (BLOCK ? x)))
```

它表示若要证明 x 的表面是一个平面，则只须证明 x 是一个长方体即可。THCONSE 函数可以将这一定理加入到定理数据库中。

再举一个稍微复杂一点的例子：

```
(THCONSE (x y z) (PART ? x ? z))  
          (THGOAL (PART ? x ? y))  
          (THGOAL (PART ? y ? z)))
```

它表示若要证明 x 是 z 的一部分，则只需证明 x 是 y 的一部分，而 y 是 z 的一部分即可。

(2) 前件定理

前件定理同后件定理正好相反，它描述的是一种前向推理，在所给定的前提下，可以得到哪些结论。例如

(THANTE (x) (BLOCK ?x))

(THASSERT (FLAT \$ x)))

它表示若 x 是一个长方体，则应该有它的表面是平面这一断言。有了上面的前件定理后，则当把 (BLOCK ABC) 这一断言加入到断言数据库之后，则 (FLAT ABC) 也将被自动地加入到断言数据库中。

(3) 删除定理

利用删除定理可以将断言从断言数据库中删除。如希望将 (BLOCK x) 从断言数据库中删除，同时希望将 (COLOR x y) 也删除，则可以写出如下的删除定理：

(THERASING (x y) (BLOCK ?x))

(THERASE (COLOR \$ x ?y)))

当有了组成 PLANNER 程序的断言和定理之后，可以通过搜索来得到对某一问题的解答，搜索是通过 THPROG 函数实现的。

例如，欲查找一块长方形的且是蓝色的积木块，则可以通过

(THPROG (x) (THGOAL (BLOCK ?x)))

(THGOAL (COLOR \$ x BLUE)))

来实现。它首先匹配目标 (BLOCK ?x)，即找一块长方形的积木，然后通过匹配目标 (BLOCK \$ x BLUE) 来看该积木是否为蓝色，若是蓝色，则找到了所要查找的积木，结束；否则继续查找其它的长方形积木，直到找到一个蓝色长方形积木块为止，或者以失败结束，说明数据库中根本就不存在蓝色长方形积木块。

在 PLANNER 中，回溯是唯一可供使用的控制结构，并且是自动实现的，不受程序员的任何控制。因这一限制，往往使得搜索在无效的进行，即使是程序员事先知道某些分支是无解的，

也不能对其加以控制，使得搜索效率极低。为此，MICRO-PLANNER 的作者之一，MIT 的 G. J. Sussman 在 PLANNER 的基础上，于1972年开发出了 CONNIVER 语言，该语言在很多方面继承了 PLANNER 的特点，如同时具有断言数据库和定理数据库等，其主要区别就是在 CONNIVER 中，不自动进行回溯，并将对问题求解的控制向用户开放，程序员可对程序流程进行明显的引导，以减轻不管对什么问题均进行盲目的 DFS（深度优先）搜索的低效率问题。

6.3 PROLOG

PROLOG 是一种基于一阶谓词的逻辑型程序设计语言，它是英文 Programming in Logic 的缩写。自从1972年法国的马赛大学首次发表 PROLOG 以来，就引起了计算机界和人工智能界的高度重视，其原因有两个方面，一方面 PROLOG 是基于一阶谓词逻辑的，而一阶谓词逻辑既有坚实的理论基础，又有较强的表现能力；另一方面是 PROLOG 具有自动推理能力，虽然这种能力目前还比较弱，但它确实由程序设计的“ How to do ”描述向“ what to do ”描述前进了一步，而这一点正是将来的程序设计语言所应具有的特点之一。

PROLOG 语言的基本组成单位是项。最简单的项是数字、常量和变量，常量用一串小写字母表示，变量用以大写字母开头的一串字符表示。例如

is-father ma append Head

如下形式的结构也是项：

<关系名> (<项1>, …, <项k>)

例如

point (X, Y, Z)

sentence (np (he), vp (v(likes, np (her))))

均是项。

表也是项， PROLOG 中的表是用符号“[”和“]”包围起来的一些元素，元素之间用“,”号分开，元素可以是常量、变量、数字或其它的表，以下都是表的例子。

[a, b, c] [1, 2, 3] []

其中，[]为空表，表示表的元素个数为零。

表的第一个元素被称为表头，而其余元素组成的表为表尾，如在表 [a, b, c] 中，a 是表头，[b, c] 是表尾。在表中可以使用结构符“|”，在“[”前面的 n 个元素是表的前 n 个元素，在“|”后面用一个表给出表的其它元素。例如，表 [a, b, c] 也可以表示为 [a | [b, c]]。利用结构符“|”可以表示出各种表的模式，如 [a | x] 表示以 a 为头的表，[H | T] 表示一个非空表。

一个 PROLOG 程序是一系列的子句，子句通常由句首和句体组成，中间用符号“:-”分开，句首一定是一个项，句体可以有几个项。如在子句

P :- Q, R, S

中，P 是句首，Q, R, S 是句体，它表示“当 Q, R, S 均为真值时，P 才为真”。

PROLOG 程序一般可分为事实和规则两部分。事实用只有句首的子句表示，并以“.”号结束。例如以下事实：

老王是小王的双亲之一

老赵是小王的双亲之一

老王是位男性

老赵是位女性

可以表示为

parent (old-wang, wang).

parent (old-zhao wang).

male (old-wang).

female (old-zhao).

一条规则可以由一个完整的子句表示，例如，规则“如果 X，Y 是 Z 的双亲，并且 X 是位男性，Y 是位女性，则 X 是 Y 的丈夫。”用子句表示出来就是

```
husband (X, Y) :- parent (X, Z),  
                  parent (Y, Z),  
                  male (X),  
                  female (Y).
```

其中句体中的“，”号表示项与项之间是“与”的关系。而子句与子句之间是“或”的关系。如关于双亲关系，我们可以定义如下：

parent (X, Y) :- mother (X, Y).

parent (X, Y) :- father (X, Y).

其含义为“如果 X 是 Y 的母亲，则 X 是 Y 的双亲之一，或者 X 是 Y 的父亲，则 X 是 Y 的双亲之一”。

PROLOG 允许递归定义，通过递归，可以定义出很多我们所需要的规则。例如，整数 N 的阶乘可以表示为

$0! = 1$

$1! = 1$

$N! = N * (N - 1)!$

若我们用 factorial (N, M) 表示 N 的阶乘为 M，则 factorial 可以定义为

factorial (0, 1).

factorial (1, 1).

factorial (N, M) :- N1 is N-1,

factorial (N1, M1),

M is N * M1.

其中 is 的功能是使得其左边的变量得到其右边的算术表达式的值。

运用表模式，我们还可以定义出许多表处理的关系。例如，假定我们用 append (L₁, L₂, L) 表示表 L 是表 L₁ 与 L₂ 合并得到的结果表。则其定义如下：

```
append ([ ], L, L).  
append ([H | T1], L, [H | T2]):- append (T1, L, T2).  
第一个子句是说，如果一个空表与表 L 合并，则结果表就是 L 本身；第二个子句是说，如果一个非空表 [H | T1] 与表 L 合并，则结果表的表头应是第一个表的表头 H，结果表的表尾 T2 是第一个表的表尾 T1 与第二个表 L 合并得到的结果表。
```

PROLOG 程序的执行是通过询问实现的，询问是通过如下形式的只有句体的子句实现的：

```
?- P1, P2, …, Pk
```

它表示“P₁ ∧ P₂ ∧ … ∧ P_k 为真吗？”其中 P_i 中可以含有变量，当询问成功时，系统给出这些变量的值。例如

```
?- append ([1, 2], [3, 4], L).  
L=[1, 2, 3, 4]  
yes
```

下面我们通过一个例子来说明 PROLOG 是如何进行工作的。假设关于 Y 是 X 的后代的子句定义如下：

```
descendant(X, Y) :- child(X, Y)  
descendant(X, Z) :- child(X, Y), descendant(Y, Z)  
也就是说，“若 Y 是 X 的孩子则 Y 为 X 的后代，或者若 Y 是 X 的孩子，而 Z 是 Y 的后代，则 Z 是 X 的后代。”，并且已知以下事实：
```

```
child(john, mary).  
child(john, bill).
```

child (bill, bob).

child (bill, susan).

这时若询问以下问题：

? -descendant (john, bob).

则 PROLOG 按自上而下的次序对子句进行搜索，寻找句首与问题匹配的第一个子句。在该例中，首先与第一个子句匹配，其结果使得 $X=john$, $Y=bob$ ，与句首匹配后，将尝试该子句的句体是否成功，因这时 X 与 Y 已有约束值，则句体变为 child (John, Bob)，由于它与任何其它的子句均不能匹配，因而失败，PROLOG 自动进行回溯，重新寻找可匹配的子句，这样就搜索到了第二个子句，并与其句首匹配，使得 $X=john$, $Z=bob$ ，而其句体成为 child (john, Y) 和 descendant (b, bob)。第一项刚好与 child (john, jiro) 匹配，从而 $Y=mary$ ，第二项变成 descendant (mary, bob)。不难得出它也不能匹配，故再次回溯，重新匹配第一项，使得第一项成为 child (john, Bill)，第二项成为 descendant (bill, bob)，它与第一个子句的句首匹配，使其句体成为 child (bill, bob)，而它刚好有与其匹配的事实，故询问得到解答，回答 yes。当询问中有变量存在时，其匹配过程与此相仿，只是当匹配成功之后，将变量的值显示出来。

在 PROLOG 中使用符号“!”（读作 cut）可以控制回溯，这对提高程序的运行效率带来很大的好处。当在询问中遇到“!”时，它象一座墙一样，阻止了 PROLOG 的回溯，使得回溯只能在它后面进行。例如，我们要查找 tom 的叔叔，可以询问：

?- father (-, tom), !, brother (-, Y).

其中符号“-”是虚变量，只起记录中间结果的作用。该询问首先查找 tom 的父亲，因每个人只有一个父亲，故不管后边匹配成功与否，都没有必要回溯对第一项寻找其它的解，这里的“!”

就起到了阻止这种无用的回溯的作用。

“!”在规则中有更强的作用，一旦“!”被求值后，它不仅起到了阻止回溯的作用，而且告诉 PROLOG 该子句是求解该问题的最后一个子句，即使还有其它的子句，PROLOG 也不尝试与它们进行匹配。

例如，我们要测试某人是否是一位男性，这有两种可能，一种可能是用事实直接记录了他是男性，另一种可能是目前没有直接记录他是一位男性，但通过记录的其它信息可判断他是否为一位男性，如看他是否为别人的父亲等。对于这两种情况，我们希望首先直接测试他是否为一位男性，当不能确定时，再通过其它信息来判断。为此，我们写出规则：

```
male-test (M) :- is-male (M) , !.  
male-test (M) :- father (M, _).
```

由于在第一个子句中有“!”，则如果能用第一个子句直接确定 M 是位男性的话，则不再使用第二个子句；只有当第一个子句匹配失败时，第二个子句才能被调用。

6.4 专家系统工具

专家系统工具是在七十年代中期开始发展起来的，它比我们通常所说的高级计算机语言，如LISP、PASCAL、FORTRAN等，在功能上又提高了一个层次，是专门用于开发专家系统的一种计算机语言。

现有的各种专家系统工具，大体上可以分为两类，一类是骨架型工具，又称为外壳，一类是语言型工具。

一般的专家系统都可以分为推理机和规则集两部分。在一个理想的专家系统中，推理机完全独立于所要求解的问题领域，系统功能上的完善或改变，只依赖于规则集的完善或改变。因此，

借用以前开发好的专家系统，将描述领域知识的规则，从原系统中“挖掉”，只保留其独立于问题领域知识的推理机部分，这样形成的工具称为骨架型工具，如EMYCIN、KAS以及EXPERT等。这类工具因其控制策略是预先给定的，使用起来很方便，用户只须将具体领域的知识，明确地表示成为一些规则就可以了，这样，可以把主要精力放在具体概念和规则的整理上，而不是象使用传统的程序设计语言建立专家系统那样，将大部分时间花费在开发系统的过程结构上，从而大大提高了专家系统的开发效率。而且这类工具往往交互性很好，用户可以方便地与之对话，并能提供很强的对结果进行解释的功能。

然而，因其程序的主要骨架是固定的，除了规则以外，用户不可改变任何东西，因而骨架型工具存在以下几个问题：

- （1）原有骨架可能不适合于你要求解的问题。
- （2）推理机中的控制结构可能不符合专家新的求解问题的方法。
- （3）原有的规则语言，可能不能完全表示所要求解领域的知识。
- （4）求解问题的专门领域知识可能不可识别地隐藏在原有系统中。

基于这些原因，使得骨架型工具的应用范围很窄，只能用来解决与原系统相类似的问题。

语言型工具与骨架型工具不同，它们并不与具体的体系和范例有紧密的联系，也不偏于具体问题的求解策略和表示方法，所提供给用户的是建立专家系统所需要的基本机制，其控制策略也不固定于一种或几种形式，用户可以通过一定手段来影响其控制策略。因此，语言型工具的结构变化范围广泛，表示灵活，所适应的范围要比骨架型工具广泛得多。象 OPS5、RLL 及 ROSLE 等，均属于这一类工具。

然而功能上的通用性与使用上的方便性是一对矛盾，语言型工具为维护其广泛的应用范围，不得不考虑众多的在开发专家系统中可能会遇到的各种问题，因而使用起来比较困难，用户不易掌握，对于具体领域知识的表示也比骨架型工具困难一些，而且在与用户的对话方面和对结果的解释方面也往往不如骨架型工具。

EMYCIN 是一个典型的骨架型工具，它是由著名的用于对细菌感染病进行诊断的 MYCIN 系统发展而来的，因而它所适应的对象是那些需要提供基本情况数据，并能提供解释和分析的咨询系统，尤其适合于诊断这一类演绎问题。这类问题有一个共同的特点是具有大量的不可靠的输入数据，并且其可能的解空间是事先可列举出来的。

EMYCIN 中，采用的是逆向链深度优先的控制策略，它提供了专门的规则语言来表示领域知识，基本的规则形式是

(IF <前提> THEN <行为> [ELSE <行为>])
当前提为真时，该规则将前提与一个行为结合起来，否则与另一个行为结合起来，并且可以用一个 -1 到 +1 之间的数字来表示在该前提下行为的可信程度。如一条判断细菌类别的规则可表示如下：

```
PREMISE: [ $ AND ( SAME CNTXT SITE BLOOD )
           (NOTDEFINITE CNTXT IDENT)
           ( SAME CNTXT STAIN GRAMNEG)
           ( SAME CNTXT MORPH ROD )
           (SAME CNTXT BURN T) ]
ACTION: (CONCLUDE CNTXT IDENT PSEUDOMONAS
          TALLY 0.4)
```

其含意如下：

如果 培养物的部位是血液
细菌的类别确不知道
细菌的染色是革蓝氏阴性
细菌的外形是杆状
病人被严重地烧伤
那么 以不太充分的证据（可信程度 0.4）说明细菌的类别
是假单菌。

在 EMYCIN 中，还提供了良好的用户接口，当用户 对系统的某个提问感到不解时，可以通过 WHY 命令向系统 询问为什么 会提出这样的问题，并且对于系统所作出的结 论，可以 通过 HOW 命令向系统询问它是如何得出这个结论的。这一点对于 诊断系统是极为重要的，用户可以避免盲目地按照系统所提供的策 略去执行。

此外，EMYCIN 还提供了很有价值的跟踪及调试 程序，并 附有一个测试例子库，这些特征为用户开发系统提供了极大的帮 助。

语言型工具中一个较典型的例子是 OPS5，它 以产生式系统 为 基础，综合了通用的控制和表示机制，向 用户 提供建立专家系 统所需要的基本功能。在 OPS 5 中，预先没有规定任 何 符号的 具体含义和符号之间的任何关系，所有 符号的含义和它们之 间 的 关系，均由 用户 所写的产生式规则所决定，并且将控制策 略 作 为 一 种 知识 对 待，同 其 它 的 领 域 知 识 一 样 地 被 用 来 表 示 推 理，用 户 可 以 通 过 规 则 的 形 式 来 影 响 系 统 所 选 用 的 控 制 策 略。

OPS 5 通 过 如 下 形 式 的 循 环 操 作 来 执 行 一 个 产 生 式 系 统：

- (1) 匹 配，确 定 哪 些 规 则 满 足 前 提。
- (2) 冲 突 解 决，选 出 一 个 满 足 前 提 的 规 则，若 没 有 一 个 满 足 前 提 的 规 则 则 停 止 执 行。
- (3) 执 行，执 行 选 定 规 则 的 动 作 部 分。

(4) 循环, 转向第一步。

这只是一个简单的控制结构轮廓, 具体的求解策略, 取决于用户使用 OPS 5 具体定义的产生式系统本身。

在 OPS 5 中, 有一个称为工作存储器的综合数据库, 它是由一组不变的符号结构组成的, 如为了表示“名字叫 H₂SO₄ 的物质是无色的而且属于酸性”, 则可以写为

(MATERIAL 'NAME H₂SO₄ 'COLOR COLORLESS
'CLASS ACID)

OPS 5 中的规则可以表示领域知识, 也可以表示控制知识, 其规则的一般形式为

(P <规则名><前提>→<结果>)

例如, 一条用于协调整体行动的规则可以如下表示, 其具体含义在右边的分号后面加以说明:

(P COORDINATE-A ; 如果有一个目标
(GOAL
 'NAME COORDINATE ; 协调系统的任务
 'STATUS ACTIVE ; 处于激活状态
 -(TASK-ORDER) ; 还没有选定顺序
 →
 (MAKE GOAL ; 则制造子目标
 'NAME ORDER-TASKS; 确定要求的顺序
 'STATUS ACTIVE ; 使其为激活状态
 (MODIFY1
 STATUS PENDING)) ; 改变其状态为挂起

OPS 5 提供了一个常规的交互式程序设计环境, 很类似于一个典型的 LISP 系统, 它允许用户跟踪或中断程序的运行来检查系统运行状态, 或在运行中改变系统等。为了在建立一些比较大

的系统时调试上的方便, OPS 5 允许通过规则名调用相应的函数, 以便检查某个应该被调用的规则为什么没有被调用, 并可以通过命令函数来查看数据库中的某些指定元素, 当系统进入不正确的状态时, 用户可以使系统后退一步, 以便查找出何处出错, 如果是因不正确的规则引起的, 可以在对规则进行修改后, 接着继续运行。

专家系统的迅速发展, 使得知识工程技术渗透到了更多的领域, 单一的推理机制和知识表示方法, 已不能胜任众多的应用领域, 对专家系统工具提出了更高的要求。因此, 近几年推出了具有多种推理机制和多种知识表示的第二代工具系统。

ART 就属于这一类系统。ART 把基于规则的程序设计、符号数据的多种表示、基本对象的程序设计、逻辑程序设计及其黑板模型, 有效地结合在一起提供给用户, 使得它具有更广泛的应用范围。并且因 ART 的组成不是对各种技术的简单罗列, 而是应用深化的集成手段有机地结合在一起的, 因此, 即便对于专门的单一应用问题, ART 也比其它任一方法单独使用或未经充分集成化的各种技术的联合使用要有效得多。

ART 的成功, 可以概括地总结为以下几个方面:

(1) ART 提供了一种知识表示语言, 可以直接表达出各种所需的事 实、规则和概念, 并且带有一个知识库编译程序, 可将多种事实、规则和概念自动地生成为过程码, 大大地加快了系统的运行速度。

(2) ART 有一个推理机来控制推理过程, 协调运行时的知识调用, 并且有对许多不同的可供选择的假设, 或时间上不同的状态同时进行推理的能力。

(3) ART 允许多种推理机制在规则上交错使用, 使得系统对各种资源的使用是合作而不是竞争。

(4) ART 提供了一个真值保持器, 通过其逻辑上的相关

性处理，使得用 ART 构造的系统可自动保持其自身的一致性。

(5) ART 提供了一个综合接口工具，通过它用户可以进行交互式图形编辑，建立分层菜单等。

6.5 小 结

1. 本章介绍了几种应用较为广泛的人工智能语言，这些语言之所以被称为人工智能语言，是因为它们能够比较方便地描述运用人工智能技术处理问题的方法。但并不是说用其它计算机语言，如 FORTRAN 语言等就不能进行描述，只不过是比起这些语言在处理符号方面要困难一些。

2. 这些语言中，最主要的语言是 LISP，它不仅是迄今为止使用最为广泛的人工智能语言，而且也是许多正在研究的高级人工智能语言的基础语言。有许多更高一层次的人工智能语言，如专家系统工具，就是在 LISP 语言的基础上实现的，因此，LISP 语言有“人工智能中的汇编语言”之称。

3. PROLOG 语言是近些年发展起来的人工智能语言，它以一阶谓词逻辑为基础，特点是用户无须考虑计算机如何求解问题，而只须把要解决的问题告诉计算机就可以了。虽然目前的 PROLOG 系统还远远达不到这一理想情况，但是它代表了计算机语言的发展方向，是一种较有前途的语言。

4. 专家系统工具是专门用来建造专家系统的更高一层次的人工智能语言。它的特点是将处理专家系统中一般问题的通用技术加以固化，使得用户可把精力集中于领域知识的整理上，从而大大缩短了专家系统的建造周期。

5. 人工智能是一门综合技术，所处理的问题非常广泛，因此任何一种语言都不可能对所有的问题处理起来都很灵活、方便，至于究竟使用哪种语言，应根据问题的性质来决定。今后人

工智能语言的发展，不可能是单一的 LISP 或 PROLOG，而是以某一语言为基础，同时提供方便而有效地调用其它语言的接口，以便更好地发挥各种语言的长处，并且在此基础上开发出使用更灵活方便的程序设计环境。

习 题

1. 分别使用 LISP、PROLOG 和 PLANNER 三种语言编写以下程序：

（1）编写计算集合的并集、交集和差集的程序。

（2）编写求一个表的深度的程序。一个表的深度定义为该表的最大嵌套层数。

（3）一个数学表达式既可以用中缀形式表示，也可以用前缀形式表示，试编写一个将它们相互转换的程序。

（4）用 A* 算法，编写一个求解八数码游戏的程序。要求该程序具有通用性，即只要给出一个问题的初始状态、目标条件、规则集和评价函数之后，就可以使用该程序进行求解。

2. 通过上述程序分析和比较各种语言的优缺点。

第七章 知识表示

知识表示是人工智能研究的一个重要课题，无论应用人工智能技术解决什么问题，首先遇到的就是所涉及的各类知识如何加以表示。不同的知识有不同的表示方法，研究知识的表示方法，不单是解决如何将知识存储在计算机中的问题，更重要的是应该能够方便且正确地使用知识。合理的知识表示，可以使得问题的求解变得容易，并且有较高的求解效率。

一个好的知识表示系统应具备以下几点：

（1）具有表示某个专门领域所需要的知识的能力，并保证知识库中的知识是相容的。

（2）具有从已知知识推导出新知识的能力，容易建立表达新知识所需要的新结构。

（3）便于新知识的获取，最简单的情况是能够由人直接输入知识到知识库中。

（4）便于将启发式知识附加到知识结构中，以便把推理集中在最有希望的方向上。

为实现以上目标，人们研究出了许多种知识表示方法，例如前面介绍的产生式就是一种最早被使用而且直到现在还被广泛使用知识表示方法。一阶谓词逻辑也是一种重要的知识表示方法，由于它具有严谨的公理体系，在机器定理证明系统中很适用。关于这两种知识表示方法，前几章已涉及到，这里不再重复，本章则主要介绍知识的几种结构化表示方法——单元、语义网络、概念从属、框架和脚本等，最后简要说明一下知识的过程式表示法。

需要指出的是，虽然目前已拥有很多种知识表示方法，但知识表示还是一个没有完全解决的问题，因而仍然是当前人工智能研究的一个重要分支。

7.1 单元表示

单元表示是从谓词演算引伸过来的一种知识表达方法，它将所有信息描述为一组单元，每个单元建立一些槽，可以通过填充槽值的办法，描述出事物之间的逻辑关系。下面给出的是一个单元的例子。

G_1

```
  self: (element-of GIVING-EVENTS)
  giver: JOHN
  recip: MARY
  obj: BOOK
```

它描述的是这样一个事件：JOHN 给了 MARY 一本书。其中 self、giver 等在“：“号左边的部分称为槽名，而(element-of GIVING-EVENTS)、JOHN等在“：“右边的部分称为槽值。其中的 self 称为伪槽，它没有具体的逻辑含义，只是说明了给东西事件 G_1 是属于给东西事件集 GIVING-EVENTS 中的一个元素。因而上例可以进一步理解为： G_1 属于一个给东西事件，该事件中的给者是 JOHN，接收者是 MARY，所给东西是书。

通过以上分析，单元表达的是在谓词逻辑中以二元谓词出现的逻辑关系。上例如用合适公式表达出来就是

$$\begin{aligned} & \text{EL}(G_1, \text{GIVING-EVENTS}) \wedge \text{giver}(G_1, \text{JOHN}) \\ & \wedge \text{recip}(G_1, \text{MARY}) \wedge \text{obj}(G_1, \text{BOOK}) \end{aligned}$$

在谓词逻辑中，可以通过简单的办法将一个多元关系转换为二元关系。实际上上述公式就是三元关系 $\text{give}(\text{JOHN}, \text{MARY}, \text{BOOK})$ 。

BOOK) 的一种二元表示。

单元表示法的主要优点是模块化, 可以很容易地添加新信息, 例如要表示出给东西事件所发生的时间, 则只须增添一个表示时间的槽就可以了, 而不必修改其它内容, 如推理系统的控制部分等。

有时槽值可以不是常量符号, 如 JOHN, 而是一个函数表达式, 特别是这个函数可能相当于另一单元的槽名。这样对于下面的事件:

John 给了 Mary 一本书。

Bill 把钢笔给了从 John 那里得到书的那个人。

我们可以用以下单元表示:

G_1

```
self: (element-of GIVING-EVENTS)
giver: JOHN
recip: MARY
obj: BOOK
```

G_2

```
self: (element-of GIVING-EVENTS)
giver: BILL
recip: recip( $G_1$ )
obj: PEN
```

在这个例子中, recip(G_1) 和 MARY 是描述同一个人的两种不同的表示方法。 G_2 中 recip 的槽值, 可以通过“计算”recip(G_1) 值, 即在 G_1 中查找 recip 的槽值得到。

槽值也可以是一个存在量词量化的变量。例如要表示句子
某人给了 Mary 一本书。

则槽 giver 的值可以用人的集合 PERSONS 中的一员的办法给出:

G_3

self: (element-of GIVING-EVENTS)

giver: (element-of PERSONS)

recip: MARY

obj: BOOK

同样，对于 JOHN、MARY 和 BOOK 等实体也可以用下述单元描述：

JOHN

self: (element-of PERSONS)

MARY

self: (element-of PERSONS)

BOOK

self: (element-of PHYS-OBJ)

我们同样可以允许在单元中有全称量词量化的变量。例如，
句子

JOHN 给每个人一样东西。

用谓词公式可以表示为

$$(\forall x)(\exists y)(\exists z)\{EL(y, \text{GIVING-EVENTS}) \\ \wedge EQ[\text{giver}(y), \text{JOHN}] \\ \wedge EQ[\text{obj}(y), z] \\ \wedge EQ[\text{recip}(y), x]\}$$

对上式进行 skolem 化，则变量 y 和 z 可用 x 的函数来代替，尤其是“给事件”y，是 x 的一个新的 skolem 函数，而不是一个常量。由 skolem 函数表示的上述“给事件”族可以用下述函数的单元描述：

$g(x)$

self: (element-of GIVING-EVENTS)

giver: JOHN

```
    obj: sk(x)
    recip: x
```

其中 obj 的槽值是 skolem 函数 $sk(x)$ ，单元族中的全称变量的辖域是整个的单元。

从以上的例子可以看出，有关集合和集合的成员关系的概念，对我们的讨论起了很重要的作用，所以一些特殊的函数对描述集合很有帮助。为了描述由一些个体构成的集合，我们使用函数 $the-set-of$ ，例如 $the-set-of(JOHN, MARY, BILL)$ 。我们还可以使用 $intersection$ 、 $union$ 以及 $complement$ 等函数来分别表示集合的交、并和补。

使用这些有关集合的函数，我们可以有效地表达析取和否定语句。例如句子

John 买了一支笔，它是钢笔或是圆珠笔，但不是红色的。
可以用下述单元表示：

B₁

```
    self: (element-of BUYING-EVENTS)
    buyer: JOHN
    bought: (element-of intersection(union(PENS, BA-
        LL-PEN), complement (RED-THINGS)))
```

而句子

John 把书给了 Mary 或者 Bill。

可以表示为

G₄

```
    self: (element-of GIVING-EVENTS)
    giver: JOHN
    recip: (element-of the-set-of (MARY, BILL))
    obj: BOOK
```

有了事件的单元表示，我们可以通过匹配的方法来得到一个

提问的解答。假设我们有以下事实单元：

G_1

```
self: (element-of GIVING-EVENTS)
giver: JOHN
recip: MARY
obj: BOOK
```

G_2

```
self: (element-of GIVING-EVENTS)
giver: BILL
recip: recip( $G_1$ )
obj: PEN
```

我们提问“Bill 给了谁钢笔？”，该提问可用如下的目标单元表示：

x

```
self: (element-of GIVING-EVENTS)
giver: BILL
recip: y
obj: PEN
```

该目标单元与事实单元 G_2 匹配的过程中将出现置换 $\{G_2/x, recip(G_1)/y\}$ ，因 $recip(G_1)$ 是一个函数，通过在 G_1 中查找 $recip$ 值来得到该函数的值，从而得到最终的置换 $\{G_2/x, MARY/y\}$ ，这样就得到了上述提问的解答。

对于谓词逻辑中的蕴涵关系，我们也可以使用单元表示。例如句子

所有计算机系的学生都已取得了毕业文凭。

其蕴含式可以写为

$$EL(x, CS-STUDENTS) \Rightarrow EQ[\text{class}(x), GRAD]$$

对于这类的蕴涵关系，可以用被称之为描绘单元的单元来表示。

一个描绘单元描述了另一个单元所标记的集合的每一个体的特性。例如，上例可以表达为下面的一个单元和为该单元的每一个体进行说明的描绘单元：

```
CS-STUDENTS
  self: (subset-of STUDENTS)
  x CS-STUDENTS
    major: CS
    class: GRAD
```

其中 $x \in CS-STUDENTS$ 的含义为 x 是一个虚拟的典型个体，其定义域是集合 $CS-STUDENTS$ 。

假设我们有事实“John是计算机系的学生。”，并想得到目标“John是一个毕业了的学生。”，首先将事实和目标表示为单元形式：

事实单元：

```
JOHN
  self: (element-of CS-STUDENTS)
```

目标单元：

```
JOHN
  class: GRAD
```

我们可以正向或逆向地使用描绘单元来达到我们的目的。当正向使用时，我们要注意 $x \in CS-STUDENTS$ 要与事实单元 JOHN 相匹配。分类的变量 x 与 $CS-STUDENTS$ 的任一元素相匹配。描绘单元应用于事实，从而对它增添了新的槽“major: CS”和“class: GRAD”，因而扩展之后的事实单元 JOHN 就与目标单元相匹配了。

当逆向使用描绘单元时，它首先应用于目标单元，从而产生子目标单元

```
JOHN
```

self: (element-of CS-STUDENTS)

由于该子目标单元正好与原始的事实单元相匹配，所以我们得出目标单元是可以证明的。

描绘单元只能描述那些可以解释成只表示有关集合成员的信息的那些蕴涵表达式，而对于其它一些较为复杂的蕴涵关系则无能为力了。为此，我们引入单元规则这一概念，每个单元规则有前项和后项两部分，且均由一些单元组成。例如，我们要表示“若 y 是 x 部门的经理，则 y 在 x 部门工作”这一规则，可以表示为如下的蕴涵式：

$$\begin{aligned} & \{EL(x, \text{DEPARTMENTS}) \wedge EQ[\text{manager}(x), y]\} \\ & \quad \Rightarrow EQ[\text{works-in}(y), x] \end{aligned}$$

用单元规则表示出来就是

R₁

ANTE: x

self: (element-of DEPARTMENTS)

manager: y

CONSE: y

works-in: x

同描绘单元一样，单元规则可以用于演绎系统。当正向使用单元规则时，如果前项中的全部单元都与事实单元匹配，则后项中的那些单元可以添加到事实单元集中去；当对一目标单元逆向使用一条单元规则时，后项单元中的一个单元必须与目标单元匹配。如果匹配成功，则已例化的前项中的单元被设置为子目标单元，然后以这些子目标单元为新的目标单元，再反复地进行逆向推理，直到所有的新的目标都与事实单元匹配为止。在对子目标单元集合（合取式）应用逆向单元规则时可能要稍微复杂一些，这与前面已经讨论过的与或图以及置换一致性测试有些类似，这里就不详述了。

7.2 语义网络

语义网络最初是在自然语言理解系统中，为表达单词的意义而设计的一种表示方法，它实际上是对知识的一种图表示法。在语义网络中，个体可以用图的一些节点表示，节点之间通过一组带有标记的弧连接，带标记的弧表达了节点之间的关系。如对于事实

知更鸟是鸟，所有的鸟均有翅膀。

为了表达知更鸟、鸟以及翅膀这三个个体，我们建立三个节点，并分别用 ROBIN、BIRD及 WINGS 表示。因知更鸟是鸟的一部分，因此在 ROBIN 和 BIRD 之间用弧线连接，并加标记 AKO (a kind of 的缩写)，以表示这种关系；而翅膀属于鸟的一个组成部分，所以在 BIRD 和 WINGS 之间也用弧线连接，并加标记 has-part，这样形成的上述事实的语义网络如图7.1所示。

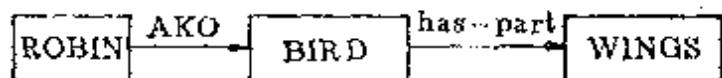


图 7.1 语义网络示例

为了增添新的事实，只需在语义网络中增加新的节点和弧线就可以了。如图7.1所示的语义网络中，增添事实

Clyde 是一个知更鸟，并且有一个叫作 NEST-1 的巢。

则图7.1变成了图7.2。Clyde是知更鸟的一个实例，因而 CLYDE

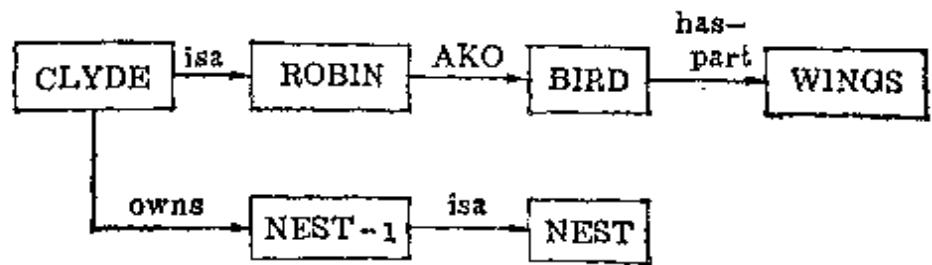


图 7.2 语义网络示例

与 ROBIN 之间用表示“是一个”的弧 isa 连接。而 Clyde 的巢 NEST-1 属于所有巢的一个实例，因而增加一个 NEST 节点，并用 isa 弧连接 NEST-1 和 NEST 节点。

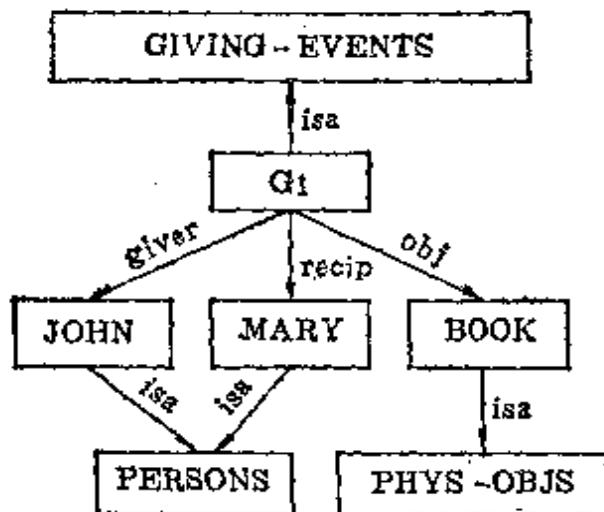


图 7.3 多元关系的语义网络

语义网络中的节点同样可以是变量，这些变量的辖域是整个语义网络。例如语句

John 给了所有人 一件东西。

可用图 7.4 所示的语义网络表示出来。其中 $g(x)$, $sk(x)$ 是 skolem 函数，而 x 是被全称量词量化的变量。

在语义网络中，连接一个节点的各个弧之间，当不加特殊说明时，表示的是“与”的关系。例如在图 7.3 中，对于节点 G_1 来说，它是一个“给事件”，并且给者是 John，接收者是 Mary，而所给的东西是 BOOK。它们必须同时成立才构成了“给事件” G_1 。

同单元表示法一样，语义网络所能表示的也是一些可用二元谓词表示的关系，如表示多元关系，则须首先将多元关系转化为二元关系，然后用语义网络表示出来，也就是在语义网络中增加一些中间节点。例如在上节中提到的事件 John 给了 Mary 一本书。可用图 7.3 所示的语义网络表示。

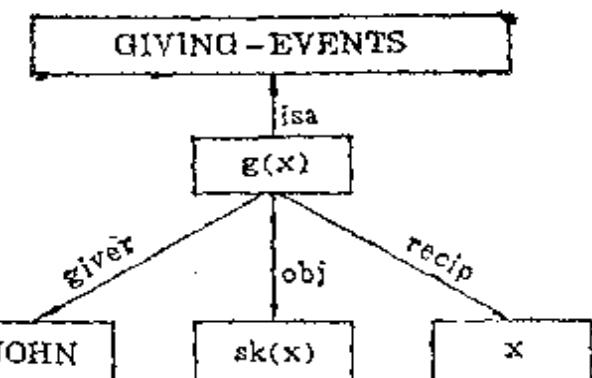


图 7.4 具有变量的语义网络

用谓词公式表示出来就是

$$\begin{aligned} &\text{isa}(\text{G}_1, \text{GIVING-EVENTS}) \wedge \text{giver}(\text{G}_1, \text{JOHN}) \\ &\wedge \text{recip}(\text{G}_1, \text{MARY}) \wedge \text{obj}(\text{G}_1, \text{BOOK}) \end{aligned}$$

为了表示“或”的关系，需要加一些特殊的标记，一种最常用的方法是将那些是“或”关系的弧用一条封闭虚线包围起来，并做标记DIS。图7.5表示的就是具有“或”关系的语义网络，其含义用谓词公式表示出来就是

$$\text{isa}(\text{A}, \text{B}) \vee \text{part-of}(\text{B}, \text{C})$$

如果“与”关系是嵌套在“或”关系内的，则这些具有“与”关系的弧用标记为CONJ的封闭虚线包围起来。例如句子

John 是一个程序员或者 Mary 是一个律师。其语义网络表示如图7.6所示。

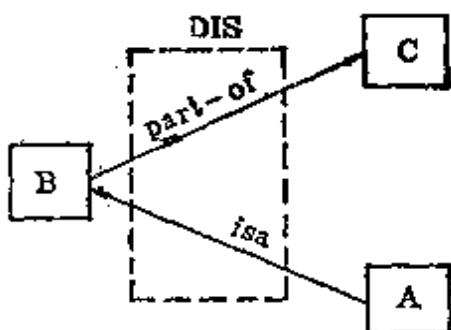


图 7.5 具有“或”关系的语义网络

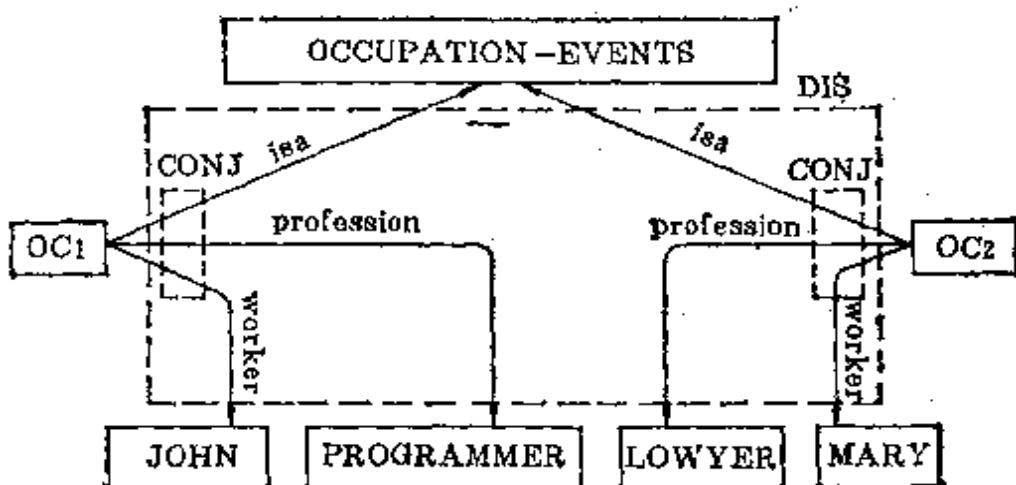


图 7.6 “或”关系中含有“与”关系的语义网络

同“与”、“或”关系一样，用带有 NEG 标记的封闭虚线也可以表示“非”关系。图 7.7 所示的语义网络表示的就是如下的“非”关系：

$$\sim[\text{isa}(A, B) \wedge \text{part-of}(B, C)]$$

同样，用标记分别为 ANTE 和 CONSE 的封闭曲线可以表示蕴涵关系，同时用一条虚线将两个封闭虚线连接起来，以表示它们是同一个蕴涵关系的 ANTE 部分和 CONSE 部分。图 7.8 就是如下句子的语义网络表示：

住在枫树街 37 号的人其职业是程序员。

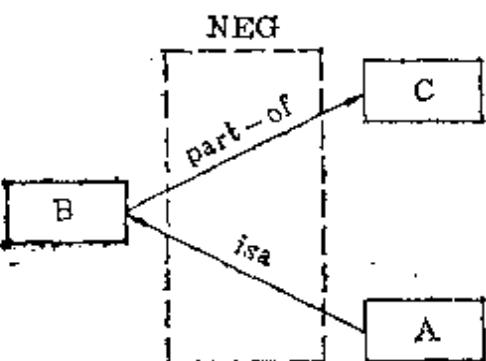


图 7.7 “非”关系的语义网络表示
同一个蕴涵关系的 ANTE 部分和 CONSE 部分。图 7.8 就是如下句子的语义网络表示：

住在枫树街 37 号的人其职业是程序员。

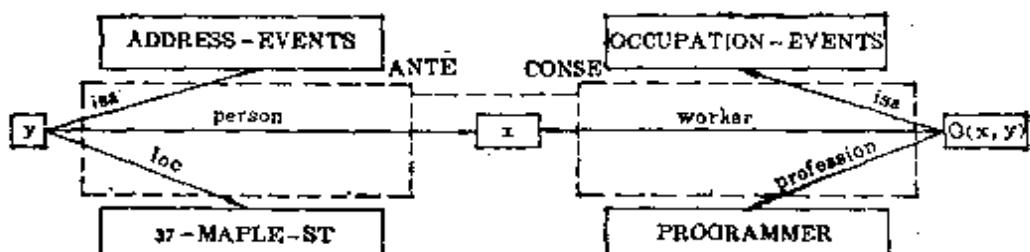


图 7.8 蕴涵关系的语义网络表示

其中， y 是一个地方，用 loc 弧表示，它的具体位置是枫树街 37 号，用 person 弧表示住在 y 处的人是 x 。 $O(x, y)$ 是一个 skolem 函数，所代表的含义是“取决于 x 和 y 的职业”，该职业的工作者用 worker 弧指示出是 x ，具体的职业内容用 profession 弧指示出是程序员。图中，用一条虚线连接了两个分别表示蕴涵关系的前件和后件的封闭虚线 ANTE 和 CONSE，以防止与其它

的蕴涵关系发生冲突。

建立好一个语义网络之后，就可以在其上面进行推理了。最简单的推理是通过继承关系来得到某些个体的一些特性值。通常这样的推理是沿着 isa 或 AKO 弧进行的。如在图 7.2 的语义网络中，通过一个 has-part 弧说明鸟是有翅膀的，这样，虽然 ROBIN 节点并没有任何弧与 WING 节点相连，但由于 ROBIN 与 BIRD 之间是用 AKO 弧连接的，因此 ROBIN 继承了 BIRD 节点的所有特性，因而 ROBIN 也就得到了有翅膀这一特性。同样，CLYDE 通过 isa 弧说明它是一个知更鸟，CLYDE 也就有了 ROBIN 节点的所有特性，因而 CLYDE 也同时获得了“有翅膀”这一特性。

运用语义网络还可以进行更复杂一些的推理，如利用含有蕴涵关系的语义网络可以进行正向和逆向推理，这里不作详述，有兴趣的读者可参考有关专著。

7.3 概念从属

概念从属是一种表达自然语言句子意义的理论，它不仅是理解语言的计算机程序基础，而且是人类语言处理的一种直观的理论。

在概念从属理论中，一个句子的表达不是建立在与句子中单词相对应的原语之上，而是建立在概念原语之上，这些概念原语的合并，就构成了单词在任一特定语言中的意义，也就是说一个句子的概念从属表达是不依赖于书写句子的源语言而存在的。

概念从属与语义网络不同，语义网络中仅提供一种结构，以任何层次表达的信息都可以放在这个结构中。而概念从属则同时提供了用来表达特殊信息块的一种结构和一组特殊原语，图 7.9 给出了一个用概念从属表达的句子的例子。

其中 ATRANS 是一个原语，它指出一种抽象关系的转移，在这

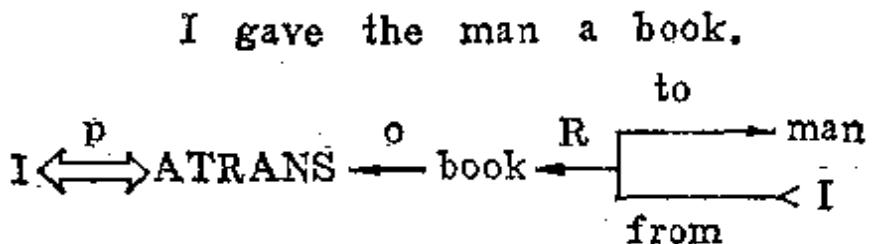


图 7.9 概念从属示例

里表示所有权的转移。单向箭头表示从属方向，双向箭头表示行为者与行为之间的双向从属关系，P 表明是过去时态，O 表明宾格关系，R 表明受格关系。

如同上例的 ATRANS 原语一样，在概念从属理论中，为了表达高级行为，使得表达所具有的含义没有二义性，提供了一组行为原语，一般包括以下十一种原语：

- (1) ATRANS 表示抽象关系的转移。如“给”。
- (2) PTRANS 表示一对象的物理位置的转移。如“去”。
- (3) PROPEL 表示把一个力作用于一个对象上。如“推”。
- (4) MOVE 表示移动自己的某一部位。如“踢”。
- (5) GRASP 表示行为者控制对象。如“抛”。
- (6) INGEST 表示动物摄取一对象进入体内。如“吃”。
- (7) EXPEL 表示动物体内排出某些东西。如“哭”。
- (8) MTRANS 表示信息的传递。如“告诉”。
- (9) MBUILD 表示根据原有信息建立新信息。如“决定”。
- (10) SPEAK 表示产生一种声音。如“说”。
- (11) ATTEND 表示将一感觉器官的注意力转向某刺激物。如“听”。

此外，还有四个概念原语：

- (1) ACT 表示一种行为。
- (2) PP 表示某一对象。
- (3) AA 表示一行为修饰语。
- (4) PA 表示一对象修饰语。

正象在图 7.9 中看到的那样，在概念从属理论中，规定了许多图示符号，来表示各个概念之间的从属关系。下面通过例子来说明各种符号规则的含义。在说明中，首先给出符号规则，然后说明其含义，最后给出一英文例句和该例句的概念从属表示。

(1) $PP \Leftrightarrow ACT$

描述行为者与它所引起的事件之间的关系，这是双向从属关系，表明行为者和事件两者所处同等地位。

John ran.

P
John \Leftrightarrow PTRANS

(2) $PP \Leftrightarrow PA$

描述对象 PP 与修饰 PP 的修饰语 PA 之间的关系。象高度这样的许多状态描述，在概念从属理论中表达为数字量。

John is tall.

John \Leftrightarrow height (> average)

(3) $PP \Leftrightarrow PP$

描述两个 PP 之间的关系，其中一个 PP 属于由另一个 PP 定义的集合。

John is a doctor.

John \Leftrightarrow doctor

(4) PP

↑

PA

描述 PP 与一属性之间的关系，说明 PP 是据有 PA 这一特性的 PP。

A nice boy.

boy



nice

(5) PP



PP

描述两个 PP 之间的关系，其中一个 PP 提供有另一个 PP 的特殊信息。按此可提供三种常用的信息类型，如所有（表达为 Poss-By）、位置（表达为 Loc）以及物理容积（表达为 CONT）。箭头方向对着正在描述的概念。

John's dog.

dog

↑ poss-By

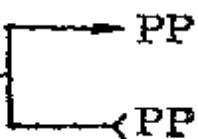
John

(6) ACT \xleftarrow{O} PP

描述行为 ACT 与其对象 PP 之间的关系。箭头指向 ACT，由特定 ACT 的上下文确定其具体含义。

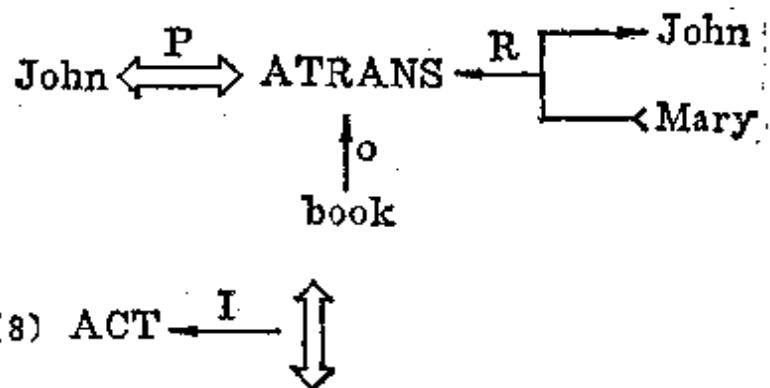
John pushed the cart.

John \xleftarrow{P} PROPEL \xrightarrow{O} cart

(7) ACT \xleftarrow{R} 

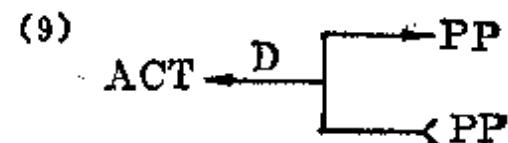
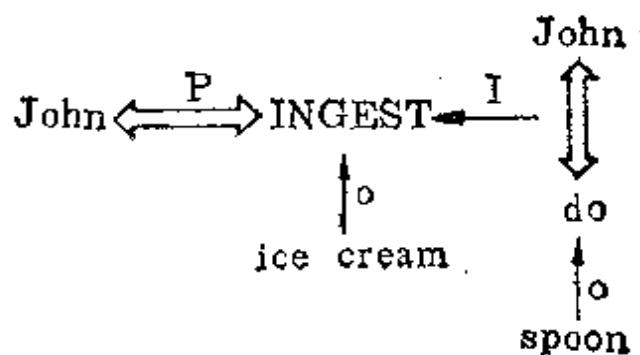
描述 ACT 与其发出处和接收处之间的关系。

John took the book from Mary.



描述 ACT 与所履行的指令之间的关系，该指令是一个含有 ACT 的概念化了的对象。

John ate ice cream.



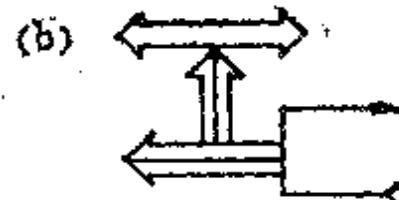
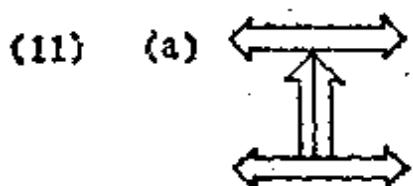
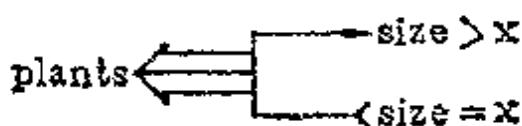
描述 ACT 与其物理资源和终极目标之间的关系。

John fertilized the field.



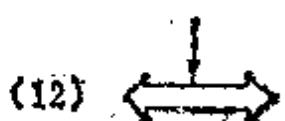
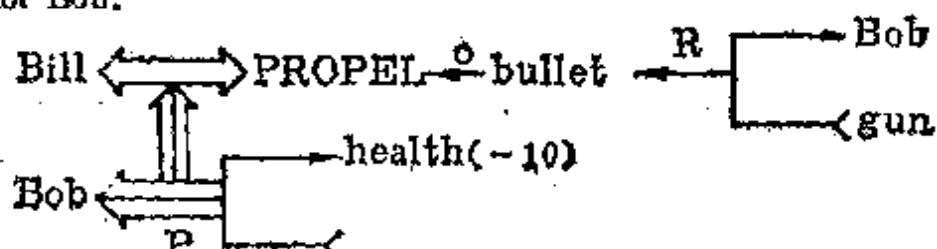
描述 PP 与一初始状态和一终极状态之间的关系。

The plants grew.



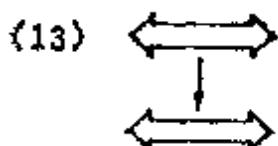
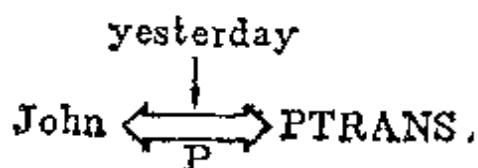
描述两个概念化了的对象之间的因果关系，注意向上的三线箭头指向的是原因，刚好与蕴涵符号相反。规则的两种形式分别表示一行为的原因和一状态变化的原因。

Bill shot Bob.



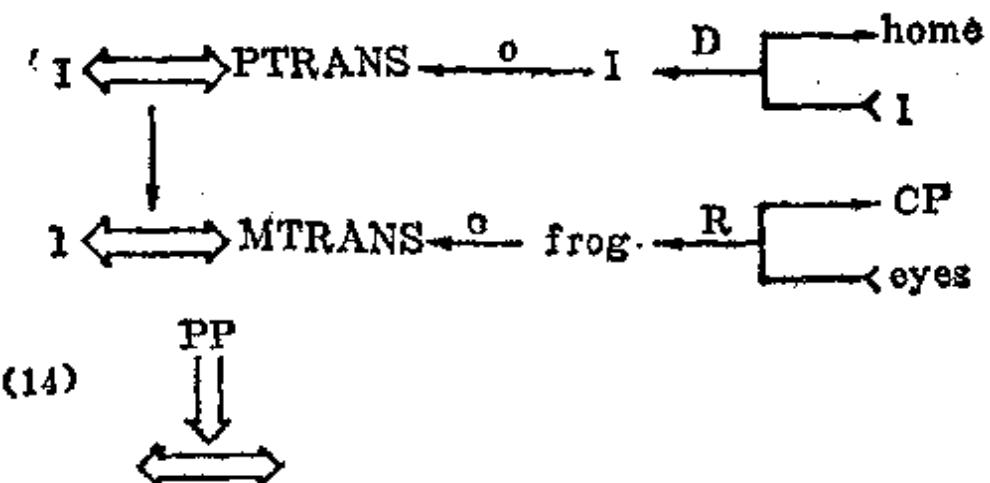
描述一概念化对象和其所表达事件的发生时间之间的关系。

John ran yesterday.



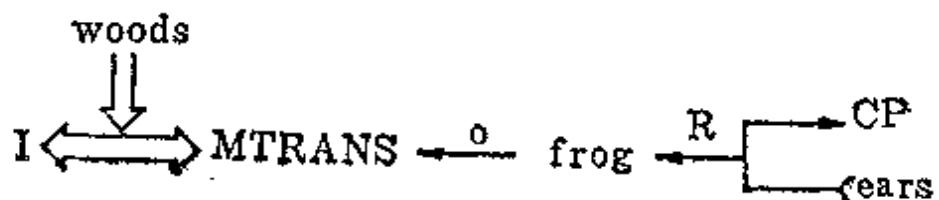
描述两个同时发生概念化对象之间的关系。通过下面的例子可以看出，概念从属效仿了人类信息处理系统的一个模式，把“看”表达为眼睛与意识处理机制之间的信息转换。

While going home, I saw a frog.



描述一概念化对象与其出现的位置之间的关系。

I heard a frog in the woods.



通过以上实例可以看出概念从属将给推理带来方便，因为好些推理实际上已经包含在表达中了。关于如何使用概念从属理论进行推理，可参阅有关文献。

7.4 框架

框架是一种关于某个体类的结构化表示法，是由 Minsky (1975) 作为理解视觉、自然语言对话和其它复杂行为的一种基础提出来的，后来逐步被发展成为一种广泛使用的知识表示方法。

框架的提出是基于这样的心理学研究成果，在人类日常的思维及理解活动中，当分析和解释所遇到的新情况时，人们并不是从头分析新情况，然后再建立描述这些新情况的新知识结构，而是使用人们从以前的实践活动中积累的知识，来联想到新情况的相应结构，并用新情况的细节装填到该结构中去。例如，当我们走进一家从来没有去过的饭店时，根据以往的经验，可以想象到在这家饭店里将看到菜单、桌子和服务员等，虽然菜单什么式样、桌子是什么颜色的、服务员穿什么衣服等细节事先并不知道，需要在进入饭店之后再仔细观察，但这样的一种知识结构事先是可以预见到的。框架就是为了在计算机中表达人们这样的知识而设计的一种组织结构。

框架通常由描述事物的各个方面的槽组成，每个槽可以拥有若干个侧面，而每个侧面又可以拥有若干个值。这些内容可以根据具体问题的具体需要来取舍，一个框架的一般结构如下：

〈框架名〉

 〈槽 1〉 〈侧面 11〉 〈值 111〉...

 〈侧面 12〉 〈值 121〉...

 ⋮

 〈槽 2〉 〈侧面 21〉 〈值 211〉...

⋮
⋮
<槽 n> <侧面 n1> <值 n11> ...
⋮
<侧面 nm> <值 nm1> ...

较简单的情景是用框架来表示诸如人和房子等事物。如一个人可以用某职业、身高和体重等项描述，因而可以用这些项目组成框架的槽，当描述一个具体的人时，再用这些项目的具体值填入到相应的槽中。图 7.10 给出的是描述 John 的框架。

JOHN

isa	:	PERSON
profession	:	PROGRAMMER
height	:	1.8m
weight	:	79kg

图 7.10 简单的框架示例

对于大多数问题，不能这样简单地用一个框架表示出来，必须同时使用许多框架，组成一个框架系统。图 7.11 所示的就是表示立方体的一个视图的框架。

图中，最高层的框架，用 `isa` 槽说明它是一个立方体，并由 `region` 槽指示出它所拥有的三个可见面 A、B、E。而 A、B、E 又分别用三个框架来具体描述，用 `must-be` 槽指示出它们必须是一个平行四边形。

为了能从各个不同的角度来描述物体，可以对不同角度的视图分别建立框架，然后再把它们联系起来组成一个框架系统。图 7.12 所示的就是从三个不同的角度来研究一个立方体的例子，为了简便起见，图中略去了一些细节，在表示立方体表面的槽中，

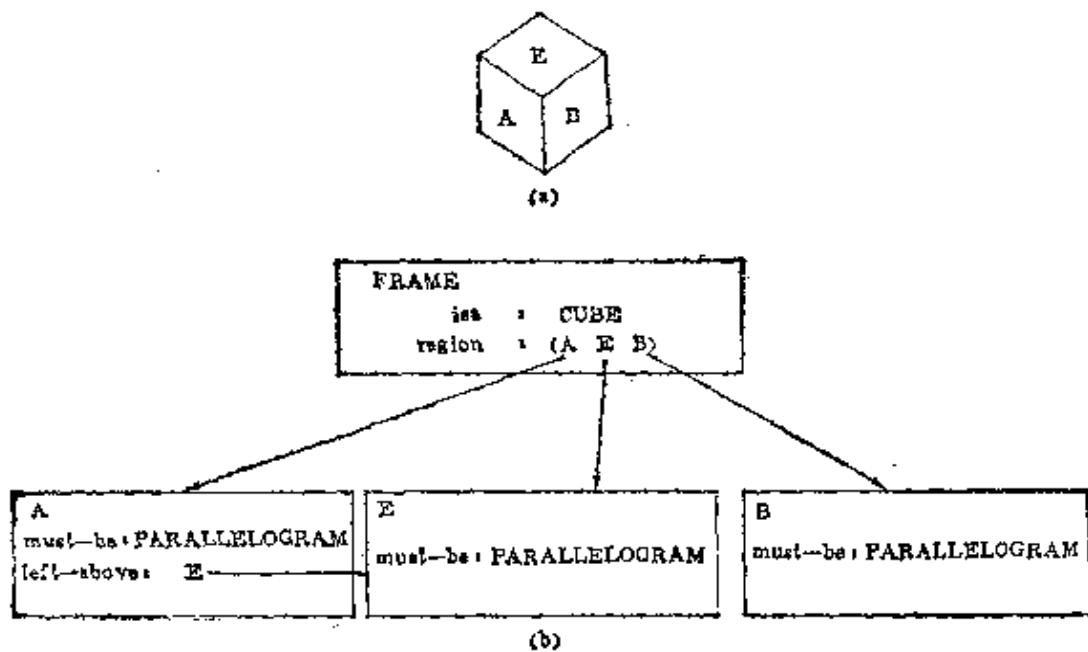


图 7.11 一个立体视图及其框架表示

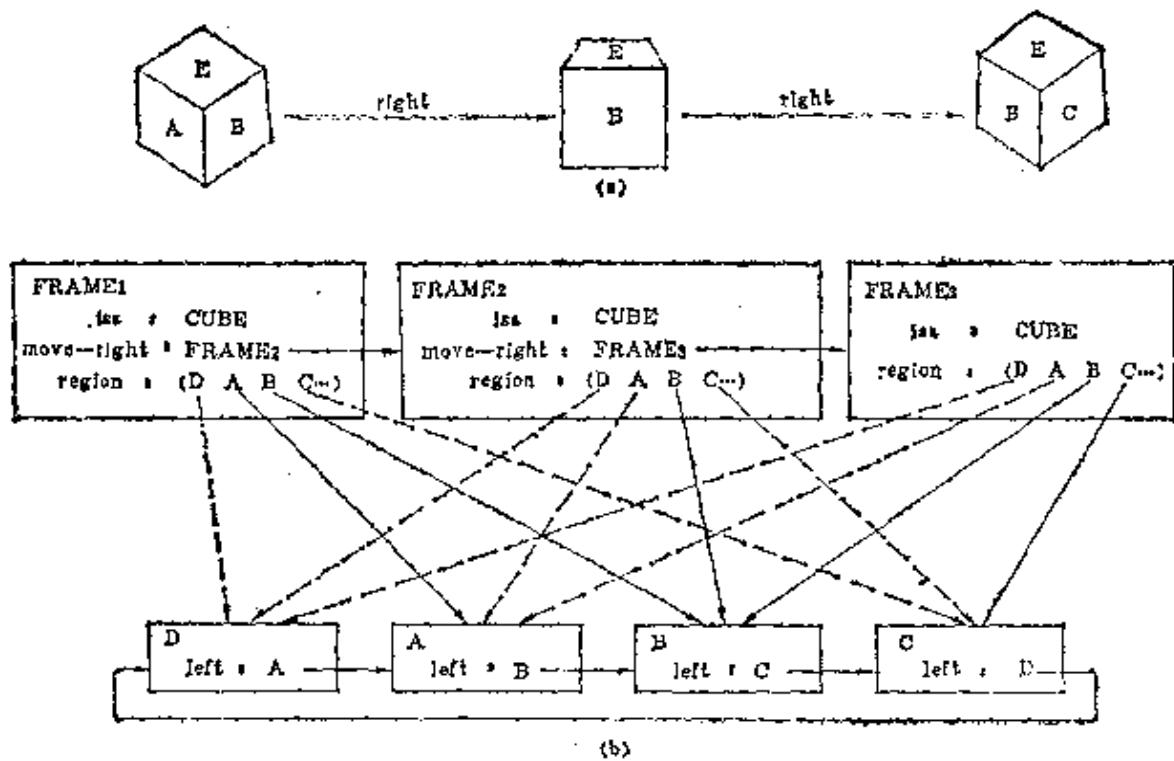


图 7.12 表示立方体的框架系统

用实线与可见面连接，用虚线与不可见面连接。

从图中可以看出，一个框架结构可以是另一个框架的槽值，
并且同一个框架结构可以作为几个不同的框架的槽值。这样，一
些相同的信息可以不必重复存储，节省了存储空间。

框架一个重要特性是其继承性。为此，一个框架系统常被表
示成一种树形结构，树的每一个节点是一个框架结构，子节点与
父节点之间用 isa 或 AKO 槽连接。所谓框架的继承性，就是当
子节点的某些槽值或侧面值没有被直接记录时，可以从其父节点
继承这些值。例如，一般椅子都有四条腿，如果一把具体的椅子
没有说明它有几条腿，则可以通过一般椅子的特性，得出它也有
四条腿。

框架是一种通用的知识表达形式，对于如何运用框架系统还
没有一种统一的形式，常常由各种问题的不同需要来决定。下面
通过一个具体的框架系统的例子来说明框架的使用问题。

该例选自“船舶积载专家系统”的知识库部分，为了便于说
明，这里进行了化简和归纳，只保留了与框架系统的一般特性有
关的部分，实际系统要比这里介绍的复杂的多。

船舶积载是船舶运输中的一个重要环节，为了保证航行安全
和货物的完好无损，在进行积载中需要掌握多方面的知识，货物
特性知识是其中的一个方面。为了描述货物的各种性质，我们建
立了一个框架系统，并对货物进行分类，连成如图 7.13 所示的
树状结构。

树的叶节点表示的是具体的货物，如花茶、绵白糖等，其它
节点是具有某些相同特点的一类事物，如花茶、绿茶都属于茶，
而茶又属于食品等。

树的每一个节点是如下形式的框架结构：

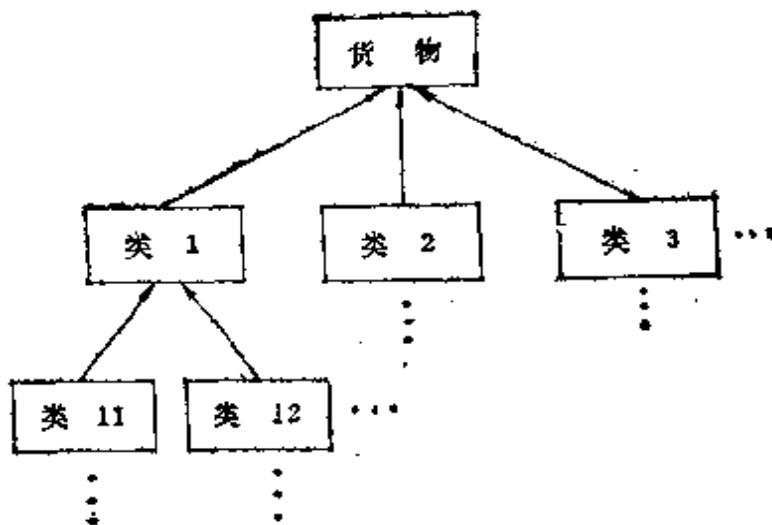


图 7.13 一个框架系统

框架名

AKO VALUE <值>

PROP DEFAULT <表 1>

SF IF-NEEDED <算术表达式>

CONFLICT ADD <表 2>

其中框架名用类名表示。AKO 是一个槽, VALUE 是它的侧面, 通过填写<值>的内容表示出该框架属于哪一类。PROP槽用来记录该节点所具有的特性, 其侧面 DEFAULT 表示该槽的内容是可以进行默认继承的, 即当<表 1>为非 NIL 时, PROP 的槽值为<表 1>, 当<表 1>为 NIL 时, PROP 的槽值用其父节点的PROP槽值来代替。其搜索过程如下:

设 F 是给定节点。

(1) 建立一个表, 初始时只有 F 一个元素。

(2) 如果表中第一个元素的 PROP 槽的 DEFAULT 侧面有非 NIL 值, 则找到了一个值。

(3) 否则, 从表中删除第一个元素, 并把第一个元素的 AKO 槽的 VALUE 侧面指向的节点加入到表的末尾。

(4) 如果找到了一个值, 则该值就是 F 节点 PROP 槽的

默认值。转(7)。

(5) 如果此时表为空，则说明 F 节点的 PROP 槽没有值。转(7)。

(6) 否则转(2)。

(7) 结束。

SF 槽记录的是该货物的积载因子 (P(重量)/V(体积))，这是一个在积载过程中可能用到的数值，IF-NEEDED 侧面说明当需要该值时，其值可以由所给的〈算术表达式〉计算出。CONFLICT 槽记录的是该类货物和哪类货物相抵，即它们不能同时放入同一个货仓中。其侧面 ADD 说明该槽值除了具有所有祖先的共性之外，还具有特殊的性质，其个性由〈表 2〉给出。其搜索过程如下：

设 F 是给定节点。

(1) 建立一个表，初始时具有 F 一个元素。

(2) 设变量 LIST 为 NIL。

(3) 取出表中的第一个元素 CONFLICT 槽 ADD 侧面的值，并将其元素加入到 LIST 中。

(4) 删去表的第一个元素，并把第一个元素的 AKO 槽的 VALUE 侧面指向的节点加入到表的末尾。

(5) 如果此时表为 NIL，则转(7)。

(6) 否则转(3)。

(7) LIST 即为 F 节点的 CONFLICT 槽的值。

以上介绍的只是利用框架的继承特点，使得存储信息减少，并且如何利用不同的侧面来得到槽的值的部分，对于运用框架系统进行推理部分，因涉及到较多领域的知识，这里就不做介绍了。

7.5 脚本

脚本是框架的一种特殊形式，它由一组槽组成，可用来描述

在特定范围的一些事件的发生序列。

一个脚本一般由以下几部分组成：

（1）进入条件

给出在角本中描述的事件发生的前提条件。

（2）角色

这是用来表示在脚本所描述的事件中可能出现的有关人物的一些槽。

（3）道具

这是用来表示在脚本所描述的事件中可能出现的有关物体的一些槽。

（4）场景

描述事件发生的真实顺序，可以有多个场景组成，每个场景又可以是其它的脚本。

（5）结果

给出在脚本所描述的事件发生以后通常所产生的结果。

下面以餐厅脚本为例说明脚本各个部分的组成。

（1）进入条件

（a）顾客饿了，需要进餐。

（b）顾客有足够的钱。

（2）角色

顾客，服务员，厨师，老板。

（3）道具

食品，桌子，菜单，钱。

（4）场景

场景 1 进入餐厅

（a）顾客走入餐厅。

（b）寻找桌子。

（c）在桌子旁坐下。

场景 2 点菜

- (a) 服务员给顾客菜单。
- (b) 顾客点菜。
- (c) 顾客把菜单还给服务员。
- (d) 顾客等待服务员送菜。

场景 3 等待

- (a) 服务员告诉厨师顾客所点的菜。
- (b) 厨师做菜。

场景 4 吃

- (a) 厨师把做好的菜给服务员。
- (b) 服务员给顾客送菜。
- (c) 顾客吃菜。

场景 5 离开

- (a) 服务员拿来帐单。
- (b) 顾客付钱给服务员
- (c) 离开餐厅。

(5) 结果

- (a) 顾客吃了饭，不饿了。
- (b) 顾客花了钱。
- (c) 老板挣了钱。
- (d) 餐厅食品少了。

一个脚本建立起来以后，如果该脚本适合于某一给定的事件，则通过脚本可以预测没有明显提及的事件的发生，并能够给出已明确提到的事件之间的联系。在使用一个脚本之前，必须将它激活，这有两种方法，方法的选用取决于该脚本在整个事件中的重要程度。

对于不属于事件核心部分的脚本，只需设置指向该脚本的指针即可，以便当它成为核心时激活，如对于餐厅脚本，在下述事

件中应采用这种方法：

苏珊在去博物馆的路上经过她喜欢的餐厅。她非常喜欢这次的毕加索作品展览会。

而对于符合事件核心部分的脚本，则应使用在当前事件中涉及到的具体对象和人物去填写脚本的槽。脚本的前提、道具、角色和事件等常能起到激活脚本的指示器的作用。

一旦脚本被激活，则可以应用它来进行推理。其中最重要的是运用脚本可以预测没有明显提及的事件的发生。例如，对于以下的情节：

“昨晚，John 到了餐厅。他订了牛排。当我要付款时发现钱已用光。因为开始下雨了，所以他赶紧回家了”。

如果提问：

“昨晚，John 吃饭了吗？”

虽然在上面的情节中并没有提到 John 吃没吃饭的问题，但借助于餐厅脚本，可以回答：“他吃了”。这是因为激活了餐厅脚本，情节中的所有事件与脚本中所预测的事件序列相对应，因而可以推断出整个事件正常进行时所得出的结果。

但是一旦一个典型的事件被中断，也就是给定情节中的某个事件与脚本中的事件不能对应时，则脚本便不能预测被中断以后的事件了。例如如下情节：

“John 走进餐厅。他被带到餐桌旁。订了一大块牛排之后，他坐在那儿等了许久。于是，他生气地走了。”

该情节中，因为要久等，所以 John 走了，这一事件改变了餐厅脚本中所预测的事件序列，因而被中断了，这时就不能推断 John 是否付了帐等情节，但仍然可以推断出他看了菜单，这是因为看菜单事件发生在中断之前。从该例也可以看出，利用脚本可以将事情的注意力集中在“因为久等，John 生气了”这样一些特殊事件的发生上。

脚本结构，比起框架这样的一些通用结构来，要呆板得多，知识表达的范围也很窄，因此不适用于表达各种知识，但对于表达预先构思好的特定知识，如理解故事情节等是非常有效的。

7.6 过程 表 示

前面所讨论的几种知识表示方法，均是对知识和事实的一种静止的表达方法，我们称这类知识表达方式为陈述式知识表达，它所强调的是事物所涉及的对象是什么，是对事物有关知识的静态描述，是知识的一种显式表达形式。而对于如何使用这些知识，则通过控制策略来决定。

和知识的陈述式表示相对应的是知识的过程式表示。所谓过程式表示就是将有关某一问题领域的知识，连同如何使用这些知识的方法，均隐式地表达为一个求解问题的过程。它所给出的是事物的一些客观规律，表达的是如何求解问题，知识的描述形式就是程序，所有信息均隐含在程序之中。因而从程序求解问题的效率上来说，过程式表达要比陈述式表达高得多。但因其知识均隐含在程序中，因而难于添加新知识和扩充功能，适用范围窄。

过程式不象陈述式那样具有固定的形式，如何描述知识完全取决于具体的问题。下面以八数码问题为例，给出一种求解该问题的过程式描述。

我们用一个 3×3 的方格阵来表示该问题的一个状态，为叙述上的方便，我们用 a~i 来标记这九个方格，如图 7.14(a) 所示。问题的目标状态设定为图 7.14(b)。当任意给定一初始状态后，求解该问题的过程如下：

- (1) 首先移动将牌，使得数码 1 和空格均不在位置 c 上。
- (2) 依次移动将牌，使得空格位置沿图 7.15(a)所示的箭头方向移动，直到数码 1 位于 a 为止。

a	b	c
d	e	f
g	h	i

(a)

1	2	3
8		4
7	6	5

(b)

图 7.14 八数码问题状态的描述及其目标状态

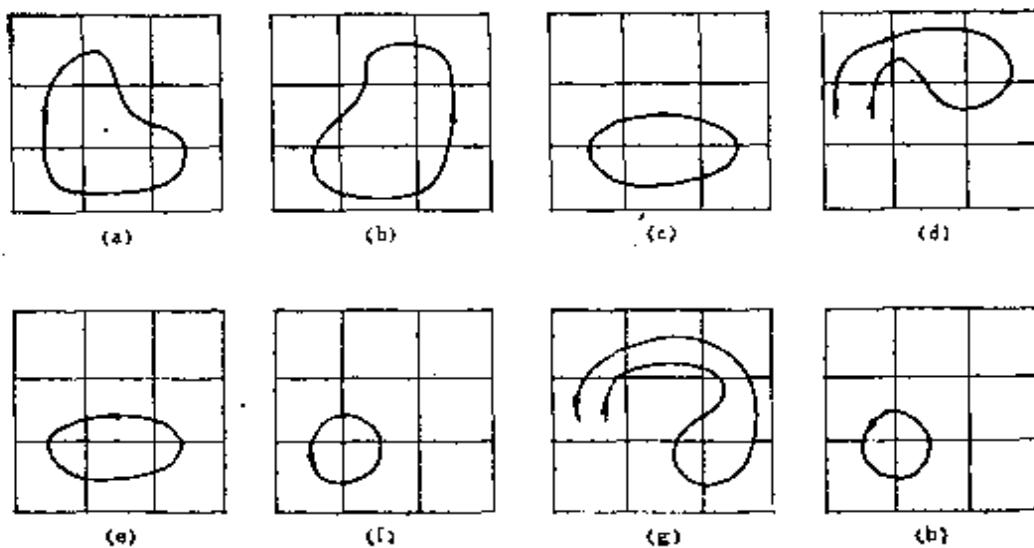


图 7.15 空格移动方向示意图

(3) 依次移动将牌, 使得空格位置沿图 7.15(b) 所示的箭头方向移动, 直到数码 2 位于 b 为止。若这时刚好数码 3 在位置 c, 则转 (6)。

(4) 依次移动将牌, 使得空格位置沿图 7.15(c) 所示的箭头方向移动, 直到数码 3 位于 e 为止。这时空格刚好在位置 d。

经过以上四步, 得到的状态如图 7.16 所示。其中“×”表

示除空格以外的任何将牌。

1	2	x
	3	x
x	x	x

图 7.16

1	2	3
	x	x
x	x	x

图 7.17

(5) 依次移动将牌，使得空格位置沿图 7.15(d)所示的箭头方向移动，直到空格又回到了 d 为止。此时状态如图 7.17 表示。

(6) 依次移动将牌，使得空格位置沿图 7.15(e)所示的箭头方向移动，直到数码 4 在位置 f 为止。若这时刚好数码 5 在位置 i 则转 (9)。

(7) 依次移动将牌，使得空格位置沿图 7.15(f)所示的箭头方向移动，直到数码 5 位于 e 为止。这时空格刚好在位置 d。

(8) 依次移动将牌，使得空格位置沿图 7.15(g)所示的箭头方向移动，直到空格又回到位置 d 为止。

(9) 依次移动将牌，使得空格位置沿图 7.15(h)所示的箭头方向移动，直到数码 6 在位置 h 为止，若这时数码 7、8 分别在位置 g 和 d，则问题得解，否则，说明由所给初始状态达不到所要求的目标状态。

图 7.18 给出了应用以上过程求解一个具体的八数码问题的例子，其中 (1) ~ (9) 九个状态分别对应了以上过程的 (1) ~ (9) 九个步骤结束时所达到的状态。

从图 7.18 可以看出，这样得到的解路显然不是最佳的，但是按这样的一种过程编写的计算机程序具有非常高的求解效率。

2		1
4	6	5
3	7	8

(0)

2	1	5
4		8
3	7	8

(1)

1	6	5
	8	7
2	4	3

(2)

1	2	8
4		6
3	7	5

(3)

1	2	8
	3	4
7	5	6

(4)

1	2	3
	4	8
7	5	6

(5)

1	2	3
7		4
5	6	8

(6)

1	2	3
	5	4
6	7	8

(7)

1	2	3
	7	4
8	8	5

(8)

1	2	3
8		4
7	6	5

(9)

图 7.18 八数码问题示例

7.7 小 结

1. 知识表示是人工智能领域研究的一个重要分支，是一个到目前为止还在继续研究的问题。运用知识求解任何问题，遇到的首要问题就是知识如何在计算机内部加以表示，知识表示的好坏，直接影响到问题求解的效率和水平。

2. 知识表示的方法虽然很多，但总体上可以分为两类，即陈述式表示和过程式表示。陈述式表示强调的是事物所涉及的对象，与事物有关的各种概念及事物之间的相互关系等，是对事物拥有知识的静态描述。其特点是知识表示直观，可读性强，模块化好，容易修改和添加新的知识等。过程式表示强调的是事物间的客观规律及求解问题的方法，与求解问题有关的所有知识，均隐含地表达为一个求解问题的过程。其特点是求解效率高，但不易修改和添加新的知识。

3. 本章主要以单元、语义网络、概念从属、框架和脚本为例，介绍了知识的结构化表示方法。这些方法之间虽然具有较大的差别，但均共享这样一个概念，即把复杂的实体描述为一组属性和关联值。这样做的好处是明显的，因为用于人工智能的许多程序设计语言都提供了实现它们的机制，从而使得这些方法能够用计算机语言加以描述，并使得建立在这些结构上的各种推理方法得以实现。

习 题

1. 把下列语句表示成语义网络的描述：

- (1) All man are mortal.
- (2) Every cloud has a silver lining.

(3) All roads lead to Rome.

(4) All branch managers of G-TEK participate in a profit-sharing plan.

(5) All blocks that are on top of blocks that have been moved have also been moved.

2. 说明下列句子的概念从属表示:

John begged Mary for a pencil.

运用此表示如何才能回答问题:

Did John talk to Mary?

3. 按电影观众的观点构造一个去电影院看电影的脚本。

4. 构造一个描述房间的框架系统。

5. 用语义网络分类体系来表达搜索技术的中心思想 (例如可以先把搜索技术划分为无信息的和启发式的) , 在你的网络中对表示的每一个集合都要有一个描述。

第八章 自然语言理解

8.1 引言

自然语言理解是人工智能领域的一个重要分支，也是一个极其活跃的研究领域。早在四十年代计算机刚刚出现时，就已经有人考虑到了将计算机应用到语言学的研究中来，并导致了“计算语言学”这一边缘学科的诞生。但由于受计算机功能及程序设计环境上的限制，当时只是进行一些编纂词条索引和词语统计等方面的工作。随着计算机技术的迅速发展，由于计算机所具有的高速的符号处理能力，很快使人们认识到计算机能够胜任远比用来计数和重排数据等工作有效得多的语言学功能。在1949年，美国的工程师 W. Weaver 首先提出了使用计算机有可能“解决世界范围内的翻译问题”，最初的想法是运用“查字典”的方法，为源语言中的每一个词汇查找出一个等价的目标语言词汇，然后再按照目标语言的语法规则，编排每一个所得词汇，从而达到翻译的目的。然而这种比较简单的思想，在实际工作中却遇到了许多意想不到的问题，正因为这些问题的出现，提醒并使得人们认识到，单纯地依靠“查字典”的方法不可能解决翻译问题，一个词可能代表了很多不同的意思，只有在理解的基础上，才能做到真正的翻译。

那么什么是“理解”呢？如何判别计算机是否理解了一句话或是一篇文章呢？这是一个目前还在探讨的问题，根据所述问题的角度不同，可以有不同的解释。从微观上来说，理解是指从自然语言到机器内部表示之间的一种映射，从宏观上讲，理解是指能

够完成我们所希望的一些功能，对此，美国认知心理学家 G.M. Ulson 曾提出了四条判别标准：

- (1) 能够成功地回答和输入材料有关的问题。
- (2) 能够具有对所给材料进行摘要的功能。
- (3) 能用不同的词语叙述所给材料。
- (4) 具有从一种语言转译成另一种语言的能力。

但是，一个自然语言理解程序的好坏或成功与否，是很难用一个绝对的判断标准来衡量的，因为理解本身就不一定是一个绝对的概念。正如象一个人听到别人对他说：“请搬一把椅子过来”。他能够马上去找椅子那样，假如你对一个航班数据库系统询问：“我想尽快地飞到上海”，如果该系统能够马上去查找最早去上海的一个航班，则同样说明它已经理解了你的问话。也就是说，“理解”也可以认为是语言到行为的一种映射。

由于存在以下三方面的问题，使得自然语言理解变得复杂起来：

- (1) 供选择的目标表示的复杂性。
- (2) 映象的类型：一对一，多对一，一对多，多对多。
- (3) 源表达中各元素间的交联程度。

第一点是说，对于源语言的目标表达，即机器的内部表示，正象在上一章中所见到的那样，具有多种不同的形式，其中既有简单的表达，又有复杂的表达，二者的表达难度相差甚远。其原因就是一个复杂表达不仅要从输入语句中获取比简单表达多得多的信息，而且其信息的获得常常要用到和语句所描述的客观世界有关的额外知识。

例如，假设我们用英语句子同一个基于关键字匹配的数据检索系统进行会话，则句子

I want to read all about last presidential election.
只须转换成如下的表达形式就可以了：

(SEARCH KEYWORDS=ELECTION \wedge PRESIDENT)

但是，如果我们是在和一个用来记载事件并且能回答关于这些事件及其相互关系的各种问题的程序进行会话，若要表达出我们所叙述的事件，则要比上例复杂多了。例如，对于下述事件：

Bill told Mary he would not go to the movies with her. Her feelings were hurt.

可以用图8.1所示的概念从属表达。显然，在转换为概念从属形式的表达过程中，增加了许多在事件叙述中没有直接描述到的额外信息。

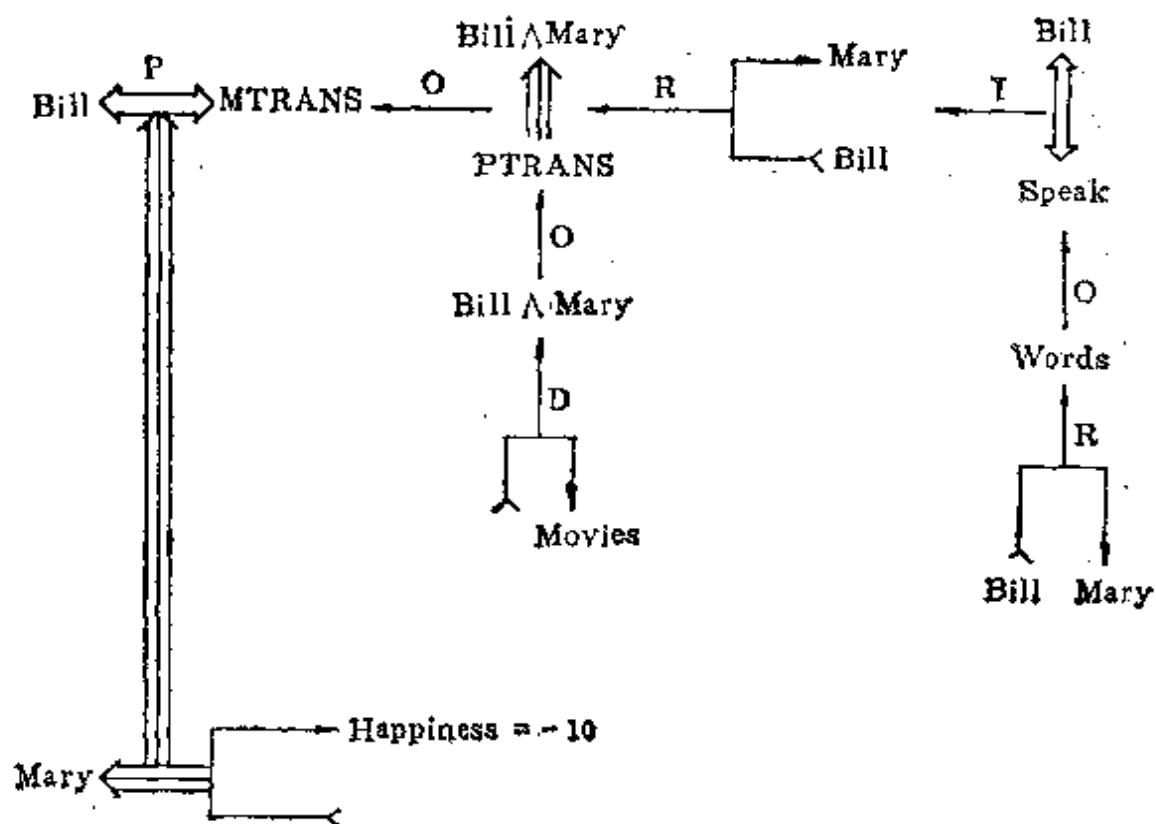


图 8.1 一段话的概念从属表示

第二点是说，一段自然语言与其所代表的含义之间，并不是

一一对应的关系，一段话可以只表达一个含义，这就是所说的一对一映射，也有可能可以理解为几个不同的含义，即一对多映射，或者几种不同的说法表达的是同一个意思，即多对一映射。

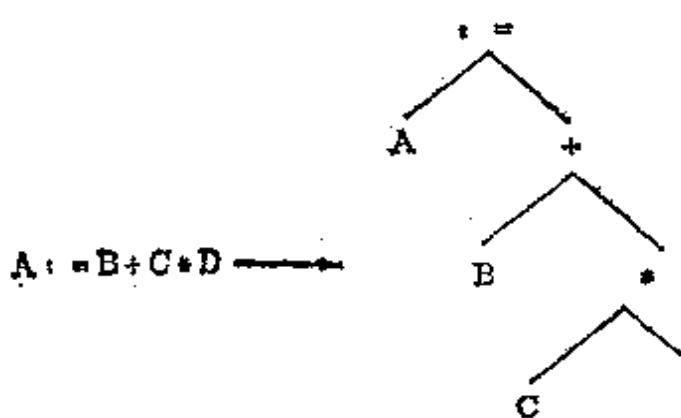


图 8.2 算术表达式语句到一个
二叉树的映射

虽然一对一映射解决起来最容易，但是它们在生动的语言中是难以找到的。作为一个例子，我们来看一下程序设计语言中的算术表达式语言，在这样的语言中，一个算术表达式语句与一个二叉树之间几乎是一对一映射，如图

8.2所示。

多对一映射最为普遍，这是由自然语言丰富的词汇决定的。特别是当将一个自然语言句子转换为一种简单表达时更是如此。例如，对于关键字数据检索系统，以下三语句含义是一样的，全都映射为同一种简单表达形式：

Tell me all about the last
presidential election.

I'd like to see all the stories

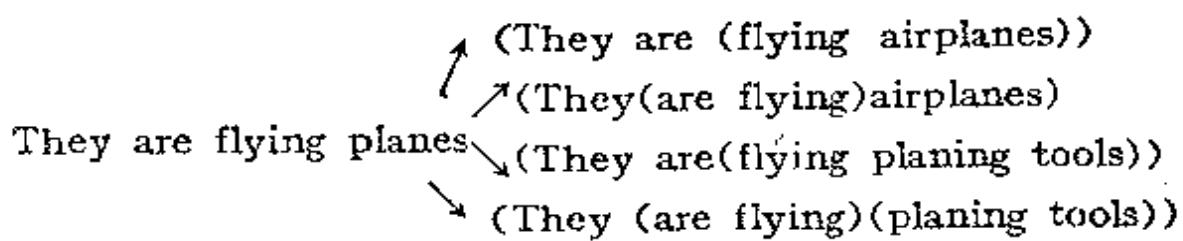
SEARCH

on the last presidential election. → KEYWORDS=

I am interested in the last presidential election.

→ ELECTION &
PRESIDENT)

一对多映射是由自然语言的多义性造成的，到底真正含义是什么，由其所处环境和上下文之间的联系确定。因而对于这些语句要得到正确的表达，需要大量的语言学以外的知识。下面给出的是一个一对多映射的例子。



第三点说的是组成语句的各元素（如词汇、符号等），它们之间的相关性如何。映射的复杂性是随着元素间相关性程度的提高而提高的。最简单的情况是语句中每一个元素的映射都与其余元素无关，如在程序设计语言中，若改变语句中的一个字，只须改变相应二叉树中的一个节点就可以了，如图8.3所示。但对于自然语言中的许多语句，要改变一个词，可能要涉及到整个句子结构的改变，图8.4给出了这方面的例子。

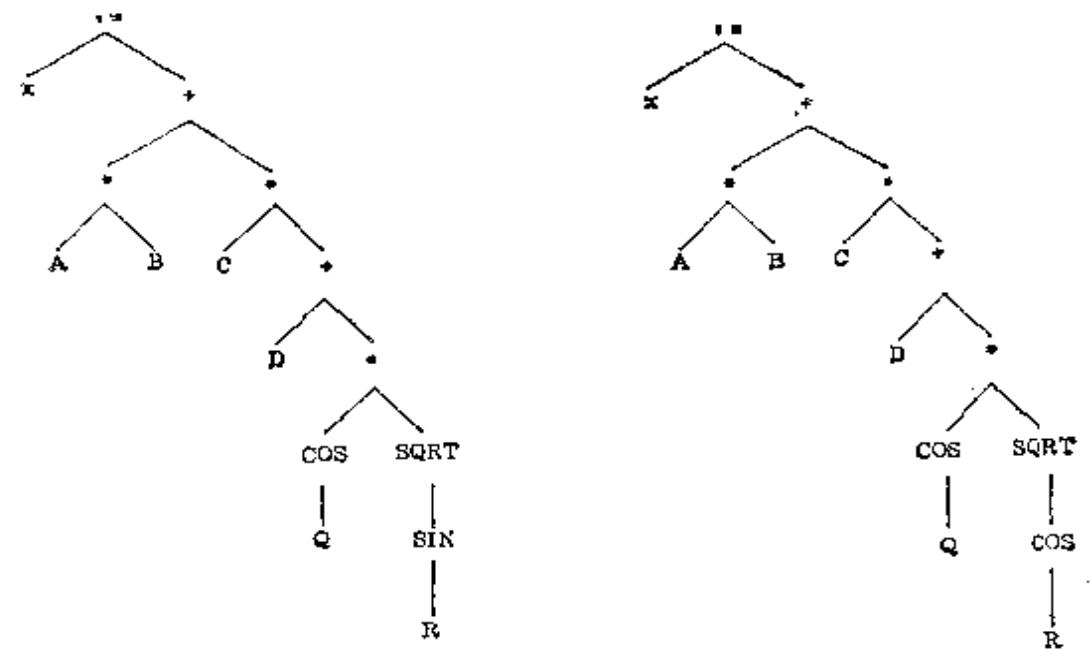
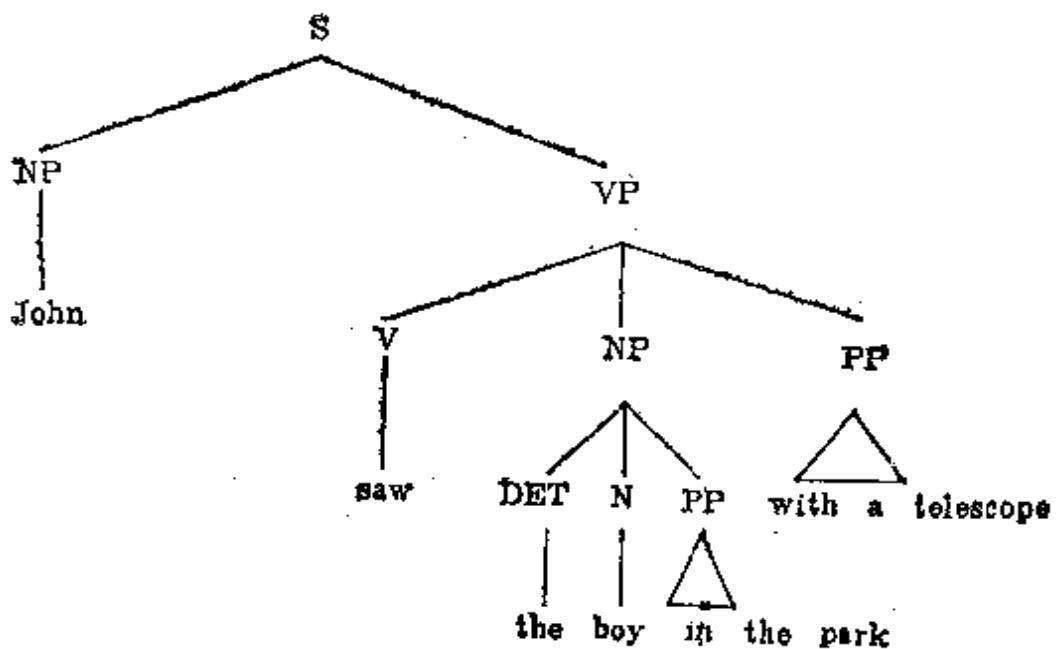
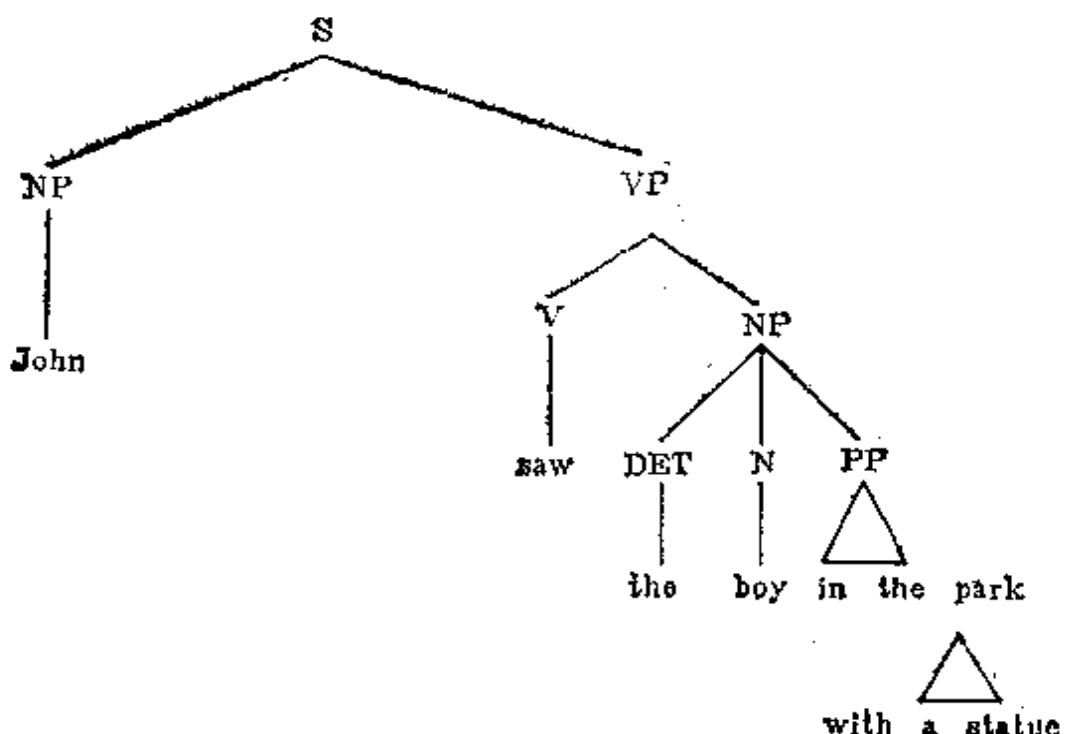


图 8.3 元素间相关性较小的例子



(a) John saw the boy in the park with a telescope



(b) John saw the boy in the park with a statue.

图 8.4 元素同相关性较大的例子

综上所述，自然语言理解是一个相当困难的问题，目前所做的工作，大多是在相当窄的一个范围内来考虑理解问题。本章仅从自然语言理解所涉及的几个方面，简单地介绍一下解决问题的几个基本方法。

8.2 简单句理解

由于简单句是可以独立存在的，因而为了理解一个简单句，即建立起一个和该简单句相对应的机内表达，需要做以下两方面的工作：

- (1) 理解语句中的每一个词。
- (2) 以这些词为基础组成一个可以表达整个语句意义的结构。

第一项工作看起来很容易，似乎只是查一下字典就可以解决。而实际上由于许多单词有不止一种含义，因而往往是只由单词本身不能确定其在句中的确切含义，需要通过语法分析、上下文关系等才能最终确定，例如，单词 diamond 有“菱形”、“棒球场”、和“钻石”三种意思，在语句

John saw Susan's diamond shimmering from across the room.

中，由于“shimmering”的出现，则显然“diamond”是“钻石”的含义，因为“菱形”和“棒球场”都不会闪光。再如在语句

I'll meet you at the diamond.

中，由于“at”后面需要一个时间或地点名词作为它的宾语，因而显然这里的“diamond”是“棒球场”的含义，而不能是其它含义。

第二项也是一个比较困难的工作。因为要联合单词来构成表示一个句子意义的结构，需要依赖各种信息源，其中包括所用语言的知识、语句所涉及领域的知识以及有关该语言使用者应共同

遵守的习惯用法的知识。由于这个解释过程涉及到许多事情，因而常常将这项工作分成以下三个部分来进行：

(1) 句法分析 将单词之间的线性次序变换为一个显示单词如何与其它单词相关联的结构。

(2) 语义分析 各种意义被赋予由句法分析程序所建立的结构，即在句法结构和任务领域内对象之间进行映射变换。

(3) 语用分析 为确定真正含义，对表达的结构重新加以解释。

实际上这三个阶段之间是相互关联的，总是以各种方法相互影响着，尽管在某种程度上把它们分开是有效的，但绝对分开是不可能的。

1. 关键字匹配

最简单的自然语言理解方法，也许要算是关键字匹配法了，在一些特定场合下是有效的。其方法简单归纳起来是这样的：在程序中规定匹配和动作两种类型的样本，然后建立一种由匹配样本到动作样本的映射。当输入语句与匹配样本相匹配时，就去执行相应样本所规定的动作，这样从外表看来似乎机器真正实现了能理解用户问话的目的。例如在一个列车运行数据库系统中，规定了以下几个匹配样本：

- (1) 从<处所>到<处所>有<车种>吗？
- (2) 从<处所>到<处所>有<? 数量><车种>？
- (3) 从<处所>到<处所>有<? 指数量><车种>？
- (4) <车次>在<处所>停吗？
- (5) <车次>经过<处所>吗？
- (6) <车次>有<车组>吗？
- (7) 到<处所>的<车种>都有<车组>吗？
- (8) <车次><? 原因>没有<车组>？

- (9) <车次><? 原因>有<车组>?
- (10) <车次><? 时刻>从<处所>开?
- (11) <车次><? 时刻>到<处所>?
- (12) 从<处所>到<处所><? 指数量><车次>最快?

其中，<…>可与任何具有规定特性的单词匹配，如<处所>可以和“北京”、“上海”等表示地点的单词匹配；<车种>可以和“特快”、“直快”等匹配；<? 数量>可与“几趟”等匹配，<? 指数量>可与“哪几趟”等匹配；<车组>可与“餐车”、“卧铺”等匹配，<? 原因>可与“为什么”、“怎么”等匹配，<? 时刻>可与“什么时候”“几点”等匹配。

如果你输入

从北京到上海有特快吗？

该语句刚好与第一个匹配样本相匹配，从而系统也就“理解”了你的问话，并去检索数据库，查看从北京到上海是否有特快，然后给出回答。

这种关键字匹配的方法，在类似的数据咨询系统中作为自然语言接口，显得特别有效，虽然它不具有任何意义上的理解。

2. 句法分析

关键字匹配法虽然简单，但却忽略了语句中的大量信息，为确保语句含义的细节不被忽略，必须确定其语句结构上的细节，这就是要进行文法分析。为此，必须首先给出说明该特定语言中字符串结构的文法，以便为每个符合文法规则的语句产生一个被称为文法分析树的结构。

关于文法的形式，在许多自然语言处理程序中提出过很多各不相同的定义，作为一个例子，下面我们给出一种文法的形式化定义。

文法 G 在其形式上为如下的四元组：

$$G = (V, \Sigma, P, S)$$

其中, V 为有穷非空集, 称作总词汇表; Σ 为 V 的一个非空子集, 称作终结字母表, 而 $N = V - \Sigma$ 称作非终结字母表; P 为如下形式的有穷产生式集:

$$\alpha \rightarrow \beta$$

式中, $\alpha \in V^* NV^*$, $\beta \in V^*$, * 表示它前面的字符可以出现任意次; S 为非终结字母表的一个元素, 称为起始符。下面给出的是一个英语子集的简单文法:

$$S \rightarrow NP VP$$

$$NP \rightarrow \text{the } NP\sharp$$

$$NP \rightarrow NP\sharp$$

$$ADJS \rightarrow \in | ADJ \ ADJS$$

$$VP \rightarrow V$$

$$VP \rightarrow V \ NP$$

$$N \rightarrow Joe | boy | ball$$

$$ADJ \rightarrow little | dig$$

$$V \rightarrow hit | ran$$

其中, 大写的是非终结符, 而小写的是终结符, \in 表示空字符串。

图 8.5 是使用该文法对语句

Joe hit the ball.
进行句法分析而建立的文法分析树。

使用给定文法, 对输入语句进行分析找到一个文法分析树的过程, 可以看成是一个搜索过程。为实现该过程, 可以使用自顶

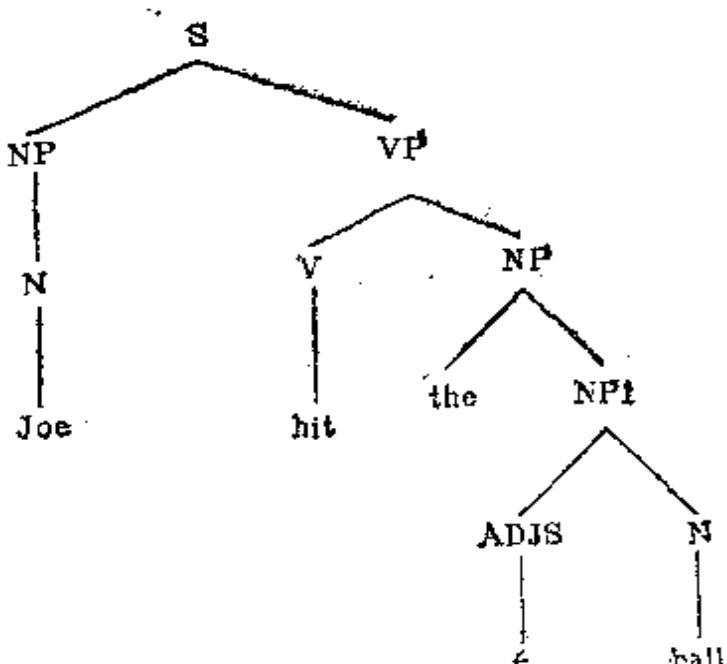


图 8.5 文法分析树示例

向下的处理方法，这和正向推理有些相象，它首先从起始符开始，然后应用 P 中的规则，一层一层地向下产生树的各个分支，直到一个完整的句子结构被生成出来为止。如果该结构与输入语句相匹配，则成功结束；否则，便从顶层重新开始，生成其它的句子结构，直到结束为止。也可以使用自底向上的处理方法，这和逆向推理有些相象，它以输入语句的词为基础，首先从 P 中查找规则，试图把这些词归并成较大的结构成分，如短语或子句等，然后再对这些成分进行进一步的组合，反向生成文法分析树，直到树的根节点是起始符为止。

不管使用哪种处理方法，都要遇到象词性选择这类的问题，比如 can 这个词，既可以是助动词，又可以是名词，对于这样的从多重选择中作出判断的问题，可以使用与搜索过程相似的控制策略，比如使用回溯策略，可首先假定 can 是一个助动词，当在以后的分析出现矛盾时，再进行回溯，重新选择 can 的词性为名词。

3. 语义分析

只是根据词性信息来分析一个语句文法结构，是不能保证其正确性的，这是因为有些句子的文法结构，需要借助于词义信息来确定，也就是要进行语义分析。

进行语义分析的一种简单方法是使用语义文法。所谓语义文法，是在传统的短语结构文法的基础上，将 N (名词)、V (动词) 等语法类别的概念，用所讨论领域的专门类别来代替。下面给出的是为舰船管理数据库系统提供自然语言接口的示例系统中的语义文法片断：

S → what is SHIP-PROPERTY of SHIP?

SHIP-PROPERTY → the SHIP-PROP | SHIP-PROP

SHIP-PROP → speed | length | draft | beam | type

SHIP→SHIP-NAME| the fastest SHIP2 |the biggest
SHIP2 |SHIP2

SHIP-NAME→Huanghe |changjiang Jinshajiang |...

SHIP2→COUNTRY S SHIP3| SHIP3

SHIP3→SHIPTYPE LOC| SHIPTYPE

SHIPTYPE→carrier |submarine |rowboat

COUNTRY S→American |French |British |Russian |...

LOC→in the pacific |in the Mediterranean| ...

从上例可以看出，该文法使用的语义类别为 SHIP 和 LOC，而不包括文法的非终结类别，如 NP 和 VP 等。

对于语义文法的分析方法，可以使用和分析纯的文法结构相类似的方法。

以上介绍的只是处理简单句理解问题的一些较简单的方法，使用这些方法，能够解决一些实际问题，但也存有很多的不足，如关键字匹配法要遗失语句中的很多信息，语义文法由于要用语义类别来代替文法类别，从而失去了许多文法上的高度概括，从而使得规则数量庞大，导致文法分析过程变得昂贵起来。

8.3 复合句理解

正象上一节所介绍的，简单句的理解不涉及句与句之间的关系，它的理解过程是首先赋单词以意义，然后再给整个语句赋以一种结构。而一组语句的理解，无论它是一个文章选段还是一段对话节录，均要求发现句子之间的相互关系。在特定的文章中，这些关系的发现，对于理解起着十分重要的作用。

这种关系包括以下几种：

（1）相同的事物，例如

“珊珊有只红气球。莎莎想要它。”单词“它”和“红气球”指的

是同一物体。

(2) 事物的一部分，例如

“小琴翻开她刚买的书。扉页已被撕坏了。”

“扉页”指的是“刚买的书”的一部分。

(3) 行动的一部分，例如

“王刚出差去上海。他乘早班飞机动身”。

乘飞机应看成是出差的一部分。

(4) 与行动有关的事物，例如

“李明决定骑车去商店。他走到车棚，可是发现他的自行车没气了。”

李明的自行车应理解为是与他骑车去商店这一行动有关的事物。

(5) 因果关系，例如

“昨天有一场暴风雪。所以学校今天停课”。

下雪应理解为是停课的原因。

(6) 计划次序，例如

“小丽想买辆新车。她决定找一份工作干。”

小丽突然对工作感兴趣，应理解为是由她想买一辆新车，买新车需要钱而引起的。

要能做到理解这些复杂的关系，必须具有相当广泛领域的知识才行，也就是要依赖于大型的知识库，而且知识库的组织形式对能否正确理解这些关系，起着很重要的作用。

如果知识库的容量较大，则有一点是比较重要的，即如何将问题的焦点集中于知识库的相关部分。上一章所介绍的一些知识表示方法，如语义网络、脚本等将有助于这项工作的进行。

例如，我们来看一下如下的文章片段：

“接着，把水泵固定到工作台上。螺栓就放在小塑料袋中。”

第二句中的螺栓，应该理解为是用来固定水泵的螺栓，因此，如

果在理解第一句时，就把需用的螺栓置于“焦点”之中，则全句的理解就不成什么问题了。为此，我们需要表示出和“固定”有关的知识，以便当见到“固定”时，能方便地提取出来。

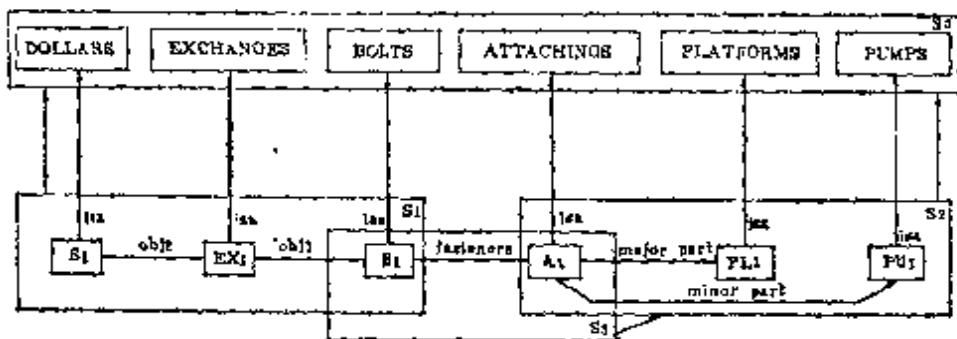


图 8.6 分区语义网络示例

图8.6给出的是一个和固定水泵有关的分区语义网络。所谓分区语义网络，就是将语义网络中的有关弧和节点集中起来，组成一个分区。图8.6所示的分区语义网络具有四个分区：S₀分区含有一些一般的概念，如美元、兑换和螺栓等；S₁分区含有与购买螺栓有关的特殊实体；S₂分区含有与把水泵固定在工作台上这一操作有关的特殊实体；S₃分区含有与同一固定操作有关的特殊实体等。运用分区语义网络，利用其分区在某些层次上的关联，可以较好地处理集中焦点的问题。当某一分区为焦点时，则其高层分区内的元素即变为可观察的了。对于上例，当第二句被理解时，因其讲的是“将水泵固定在工作台上”这一事件，因而焦点处于S₂分区。由于S₀分区的层次高于S₂分区，所以S₀分区是可以观察的。当理解第二句时，显然“螺栓”不能与S₂分区的任何元素匹配，因而焦点区由S₂变成更低一级的S₃分区，并且使得“螺栓”与B₁匹配，匹配的结果使得第二句中的“螺栓”必定是第一句中用来进行固体的螺栓，从而使得前后两个句子成为一个前后连贯的文章片断。

当输入的文章片断描述的是有关人或物的行为等情节时，可以使用目标结构的方法来帮助理解。例如

“王强很想喝汽水。他向行人打听最近的冷饮店在哪里。”

对于这样的情节，弄清楚人物的目标及其如何达到目标是理解的重点所在。在上例中，目标是得到汽水喝，为此，王强必须去冷饮店，而要去冷饮店则必须知道冷饮店的位置，为达到这一新的子目标，王强应该去询问别人。从而得出达到目标的行为规则：

询问某人→得知冷饮店→去冷饮店→买汽水→喝汽水

常见的一些目标有以下几种：

- (1) 满足式目标 如睡眠、饥饿和喝水等。
- (2) 享受式目标 如娱乐、竞赛等。
- (3) 成就式目标 如占有、权力和地位等。
- (4) 保持式目标 如健康和财产等。
- (5) 工具式目标 可作为其它更高级目标的前提的那些目标。

为了便于理解，对于这些常常出现的各种目标，可以编写好相应的规划，一旦需要时就去调用它们，这样，当情节中某些信息省略时，也可以通过这些规划推导出来。

下面给出的就是这样一个规划的例子：

```
USE(x) = D - KNOW(LOC(x)) +
          D - PROX(x) +
          D - CONTROL(x) +
          I - PREP(x) +
          DO
```

它表明如果目标是“使用 x”，则应依次满足 ① 了解 x 在什么地方，② 靠近 x，③ 控制 x，④ 使 x 成为备用，⑤ 去做。例如，

对于上例情节中的目标，可以表示为 USE(beer)，这样一来，就不难于回答诸如

“王强为什么打听冷饮店在哪儿”这样的问题了，这是由于 USE(beer)规划的第一步是“了解 beer 在什么地方”的原因。

8.4 语 言 生成

所谓语言生成，就是将在计算机内部以某种形式存放的需要交流的信息，以自然语言的形式表达出来，因而从某种意义上来说，语言生成是自然语言理解的一个逆过程。一般包括以下两部分：

(1) 建立一种结构，以表达出需要交流的信息。也就是进行“构思”，确定要“说”的内容。

(2) 以适当的词汇和一定的句法规则，将要交流的信息以句子形式表达出来。

同自然语言理解一样，语言生成的处理方法有很多种，这不仅因为它们所采用的内部表达结构不同，如采用语义网络或者概念从属等，而且因为语言生成的目的不同。如有的目的是为了对输入文章作摘要，有的是为了作为问题回答系统的人一机界面等。

语言生成也有许多难点，特别是第一部分，显得更加困难一些。有时，要交流的信息由问题回答系统在回答问题时加以估计。在这些系统中，信息的生成过程要受到回答问题的约束。

在语言生成系统中遇到的许多问题与在自然语言理解系统中所遇到的问题是一样的。例如，在自然语言理解系统中必须消除头语（字词或短语）的重复引用问题，为生成好的文章，也必须解决好这个问题。请看下面的短文：

“小明看到商店橱窗里的一辆自行车。小明想要那辆自行车。”

这篇短文看起来显得不太自然，其原因就是没有使用代词。若用代词替代已出现过的事物，则生成的短文就自然一些：

“小明看到商店橱窗里的一辆自行车，他想要它。”

但是，也不是对任何句子都可以这样简单地处理，其必要条件是所得到的句子不至于因代词的出现而产生多义性。例如，假定原文为

“小明看到闪红光的推车里的一个蓝气球。小明想要这个蓝气球。”

若简单地使用代词，则会产生

“小明看到闪红光的推车里的一个蓝气球，他想要它。”

这里的“它”就有二义性，它有可能是指气球，也可能是指小推车。

为使得短文不至于产生二义性，这里应该生成以下这样的短文：

“小明看到闪红光的推车里的一个蓝气球，他想要这个气球。”

这样得到的文章就显得既自然，又没有二义性了。

总之，语言生成需要解决几乎所有的在自然语言理解中遇到的问题，其处理方法，也可以反向地使用在自然语言理解中所使用的各种方法，进一步的介绍可参阅有关文献。

8.5 机器翻译

正象在本章的引言部分所叙述过的那样，从计算机刚刚出现，人们就想到了使用它来进行机器翻译等方面的工作。这方面较早期的工作是希望借助于字典，将源语言直接映射为目 的语言，然而这项工作最终以失败而告终。其原因是翻译必须在理解的基础上才能正确地进行，否则将会遇到一些无法克服的困难。例如：

（1）词的多义性。源语言可能一词多义，而目的语言要表

达这些不同的含义需要使用不同的词汇。为选择正确的词，必须了解所表达的含义是什么。

(2) 文法多义性。对源语言中合乎文法规则但具有多义的句子，其每一可能的意思均可在目的语言中使用不同的文法结构来表达。

(3) 头语重复使用。源语言中的一个代词可指多个事物，但在目的语言中要有不同的代词，正确地选用代词需要了解其确切的指代对象。

(4) 成语。必须识别源语言中的成语，它们不能直接按字面意思翻译成目的语言。

如果不能较好地克服这些困难，就不能实现真正的翻译。下面给出的一对句子，是说明使用查字典方法不能成功地进行翻译的典型例子：

(1) 精神是乐观的，而肌肉是软弱的。

(2) 伏特加酒是好的，而肉是腐烂的。

其中第一句是英语原句的意思，第二句是将第一句译成俄文，然后再将其译成英文后的意思。

防止这类问题出现的方法，就是首先理解源语言文章，然后再使用目的语言，按理解的含义生成出文章来。因而可以说语言理解和语言生成是机器翻译的基础，这两个难题解决了，机器翻译也就容易实现了。

8.6 小 结

1. 本章介绍了自然语言理解这一十分引人注目而又极其困难的问题。其要点是理解自然语言需要大量的知识，这不仅包括语法、使用习惯等语言学方面的知识，而且需要大量的与所述内容有关的背景知识。因此，自然语言理解与知识表示之间有着紧

密的联系，事实上第七章中介绍的许多知识表示方法，都是在研究自然语言理解的过程中发展起来的。

2. 什么叫做“理解”，这是一个目前还没有完全阐明的问题，一般地，可以认为“理解”是自然语言语句到机器内部某种表示结构的映射，运用该结构，可以进行我们希望的某些操作，如能够回答有关问题等。为了建立起这种结构，除了需要理解语句中的每一单词的正确含义外，一般还要进行句法分析、语义分析和语用分析这三方面的工作，这些工作又是相互影响的，不能绝对分开进行。

3. 语言生成可以认为是语言理解的一个逆过程。它的难点在于如何进行“构思”，即如何将要交流的信息表达为机器内部的一种结构。这里所遇到的是和自然语言理解同样的问题，许多用于自然语言理解的方法，对于语言生成同样适用。

4. 机器翻译必须在理解的基础上才能进行，单纯地依靠“查字典”的方法，不可能解决这一问题。如果语言理解和语言生成这两个问题很好地解决了，则机器翻译问题也就不难实现了。

习 题

1. 考察句子

The old man's glasses were filled with sherry.

选择单词“glasses”合适的意思要求什么信息？什么信息暗示着不合适的意思？

2. 对下列每个语句给出句法分析树。为对这些句子作出正确的句法分析，除了英语句法外，还需要哪些知识？

- (1) John wanted to go the movie with Sally.
- (2) John wanted to go to the movie with Robert Redford

(3) I heard the story listening to the radio.

(4) I heard the kids listening to the radio.

3. 考察下列句子：

Put the red block on the blue block on the table.

(1) 写出句中全部合乎句法规则的有效句法分析。

(2) 如何用语义信息和环境知识选择该命令的恰当含义？

4. 设有一学生成绩管理数据库，试写出适合于查询该数据库内容的匹配样本。

5. 利用产生式系统编写一个句法分析程序，在该系统中，状态如何表示？如何给出目标条件？

第九章 感 知

9.1 感知问题概述

客观世界的环境充满着各种事物，人们通过自己的感官——视觉、听觉、触觉、嗅觉和味觉能感知到丰富多采的客观事物的各种信息。在人工智能的研究中，需要分析来自客观世界的信息并要确定出这些信息所表达的事物，这就是感知程序要解决的问题。由于视觉和听觉是人们感官中最主要的部分，例如人们接受外界信息有70%来自视觉系统，人类之间的信息交流主要通过听觉进行，因此人工智能中对视觉和听觉已开展了广泛的研究。

视觉系统接受外部世界两维图象的信息，通过处理希望能给出一个有意义的物体描述。语音识别系统则接受语音信号（声波），通过处理和分析希望给出该信息所代表的有意义的语句结构。它们的信息处理过程实际上都是一种分类的处理，这种分类过程是分层来进行的。例如，为了识别一幅景象，先要识别各种轮廓线，然后组合这些线条表示出若干物体及其影子的形状，最后再把它们组合起来，产生出房子和场院等全貌的图象。又如为分析一个语句，首先需识别出每一个音符，然后将单个的音符组成单词，再将单词组合成有含义的句子结构。可以看出这种层次分类过程完全对应于我们所要感知的外部世界的层次结构体系。

实际上我们对某一个输入信号按层进行分类的过程，要比上面的描述复杂和困难得多，主要有几个原因：

1. 在每一个层次中进行分类的过程，往往不是相互独立的，

即在某一层中进行分类时，通常要受到上一层或下一层分类过程的影响，下面举两个简单例子来说明。

语音识别简例

假定一小段语音信号已经识别好是由如下的几个发音符号组成：

k a t s k a r s

下一个层次的分类问题是把这些音符划分为单词，可以看出至少有两种分法：

cat scares

或 cats cares

显然如果没有附加句子结构的知识（属下一个层次）的帮助，则很难确定那一个分法较为合适，因为这两种分法对不同的上下文句子都可能是正确的，例如

The cat scares all the birds away.

A cat's cares are few.

由此看出各层之间的相互作用，增加了分类的困难。

图象理解简例

图象理解问题也具有类似的二义性问题，设图 9.1 是从二维原图象中提取得到的一张线条画，下一个层次的工作是把这张画分解成若干个物体。设从图的左边开始，识别了标记为 A 的物体，是否 A 这个物体只到中间那条垂直线为止呢？显然如果不通过另一个直立积木块物体的观察，就无法决定 A 这个物体是否延伸到右半部，因此物体划分过程也涉及到物体的组合问题。

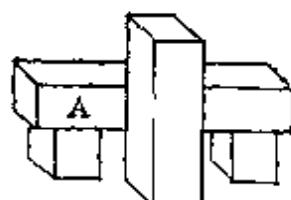


图 9.1 局部有二义性的线条画

2. 感知输入信号的许多特性是相对的，因而难以使用绝对的模式匹配技术。例如两个人的发音不可能完全等同，甚至同一

个人说一个字也不总是以同一方式来说，在图象理解中，对图象也只能作相对的度量，同一个场景，距离近和距离远摄取时，获得的图象就会有差别。

3. 在现实世界中，几乎不可能一次只感知一个单独的信号，例如语音识别问题中，目前机器可以实现以某种可接受的精度来理解单个的单词，但多个单词连读时形成的连续语音信号的识别问题就难得多。在图象理解中也类似，给出的线条画各个部分并不是完全和实际物体一一对应，一些物体有一部份被另一些物体挡住等等。此外感知的信号带有各种噪声，这都将给识别带来困难。

总之尽管研究感知问题有许多困难，但从 70 年代以来在语音识别和图象理解方面还是不断取得进展，并有一定的成果。本章仅以视觉问题中最基本的人工智能问题做一些讨论。

9.2 求解感知问题所使用的技术

不论是语音或者是图象问题的分析过程，都要把整个理解过程划分为几个处理阶段来进行，通常可分为 5 个阶段：

1. 数值化

首先要将连续的输入信息离散化，如对连续语音信号以某种采样频率（如20kHz/s）定时测量其振幅；对图象视频信号则把图象区域分解为某一固定数目的象素（如 $128 \times 128/\text{平方英寸}$ ），对黑白图象，象素值可以二值（0、1）表示，也可以多级灰度值表示，对彩色图象，则象素值表示成含有基本颜色的数量。

2. 平滑

目的是消除输入信号中有个别较大变化的数据。由于现实世

界中大多都是一些连续信号，因此输入中的这些突变式的信号通常都是由于随机噪声引起的，要通过滤波处理加以消除。

3. 分割

把由数值化产生的一些小群体组合成与信号的逻辑成份相对应的大群体，即要分割为一个个区段来处理。例如对语音理解问题，这些分割段就对应于各种发音（如 s 或 a），语音信号中的这些分割段通常称为音素；对视觉理解问题，这些分割则对应于图象中各物体所具有的某一种重要的特征，图象中的这些分割段通常由若干明显的线条组成。

4. 标记

对每一个分割段加一个标记，该标记表明这个分割段属于现实世界模块中哪一部分。例如，对语音问题，就是要在分割段标上音素的标记（如 a 或 s）；对视觉问题，则标上“该线条属于一个图形的外边界”等。通常标记过程总是不可能仅仅依靠观察就能决定某分割段应当加什么标记，因此标记过程实际上要做的事情是：

（1）对某一个分割段赋给多个可能具有的标记，等到分析阶段再根据整个输入上下文的关系，选择其中一个有意义的能代表实际情况的标记；

（2）或者标记过程能应用其自身的分析过程，对若干分割段进行检验，以便限制每一个分割段的标记选择范围。

5. 分析

把已标记的各个分割段组合起来，形成一个连贯的物体，这一步是要进行最起码的综合处理，在分析中往往要利用大量的领域专业知识，且有许多方法可用。但应当指出，几乎所有的分析

过程，其共同点都是约束满足这一基本方法的变种，这是由于只用低层次的标记来进行高层次的分析过程中，通常对给定的一个分割段，存在着许多可能的解释，但考虑到周围的其他分割段时，满足相互间都能相容解释的数量将大为减少，一个重要的原因是信号中全局作用的知识起了作用。例如语音问题中，整个语句的各种语调模式，视觉问题中利用一个光源照射，使图象出现明暗的部分，都可形成约束条件，从而限制了解释的数目。

以上几个处理过程，要使用某种结构，把它们组成为单独的系统，解决实际的感知问题。下面通过最简单的视觉问题，讨论分析阶段中约束满足法的基本思想，来说明人工智能技术的应用问题。

9.3 约束满足法

景物理解是一个复杂和困难的问题，如何利用有效的约束条件对解决这个问题有重要的作用，本节将以积木世界中线条画的景物分析问题，来讨论该问题的描述及约束满足法求解这一问题的基本思想。

积木世界景物理解的问题是将视觉信号经过处理转换为线条画，然后把线条画解释为相应的积木块，其间的关系及性质等描述。前一个问题主要涉及信号处理的领域，后一个问题则属人工智能领域。这里我们主要讨论线条画的解释问题。首先要对线条加以分类，然后按现实可能把线条组合成一定类型的结点，最后用约束满足法给出线条画中各线条的解释，从而达到对景物的理解。

1. 线条的类型

图 9.2 是一个积木世界的线条画，图中的线条有边界线、凸

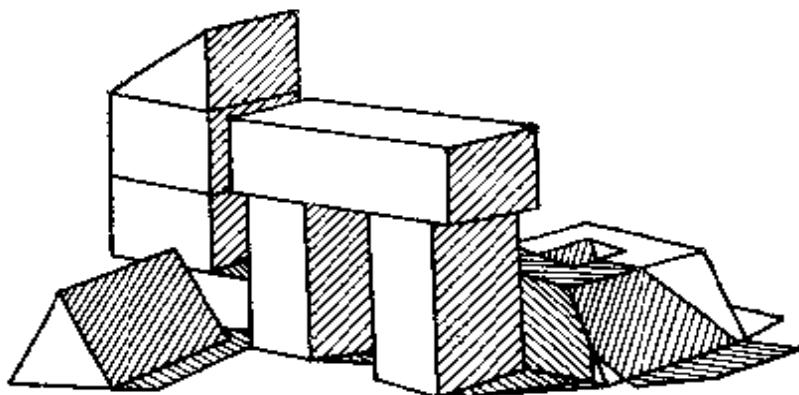


图 9.2 一个线条画

形线、凹形线、阴影线和缝隙线几种。为了进一步简化问题起见，再假定讨论场景有光照，不出现阴影和缝隙的多面体线条画。在这种情况下线条可分为边界线和内部线(凸形线和凹形线)两种，边界线划分属于不同物体的区域，内部线则分割同一物体的区域。为了能在图上表示这些线条类型，还规定相应的符号作为标记：

边界线：用箭头标记，即在线段上加上箭头>或<，其方向规定为沿着箭头方向移动时，该物体的区域处在边界线的右侧；

凸形线：用+号标记，可在线段边上标注+号表示；

凹形线：用-号标记，可在线段边上标注-号表示。

图 9.3 是一个实心角尺线条画按这种规定给出的线条标记，

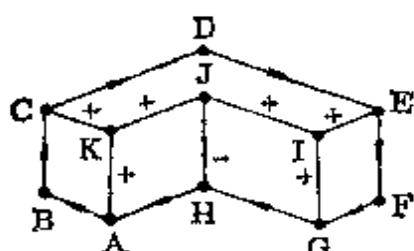


图 9.3 角尺线条画的线条类型标记

这里是按人们对这个物体状态的理解给出对线条的标记。反过来如果一个线条画每根线条都给出合理的标记，自然也能识别出这个线条画所代表的具体物体。下面我们先对这些线条所组成的结点类型进行分析，然后再来讨论

线条如何标记的问题。

2. 结点类型的分析

在一般情况下，线条画中各线条相交时形成的结点类型如图9.4所示，为了进一步简化讨论，除了没有阴影和缝隙的假定外，再假定所有的结点都限于物体三个平面交汇构成的结点（即不出现如图9.5所示的情况），以及要选择一般的观察点，使所得到

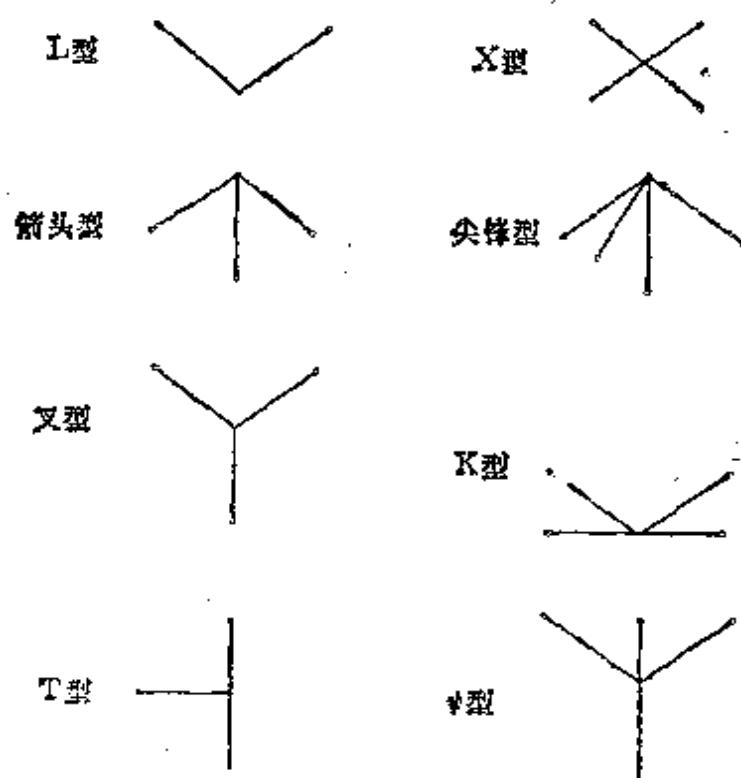


图 9.4 结点类型

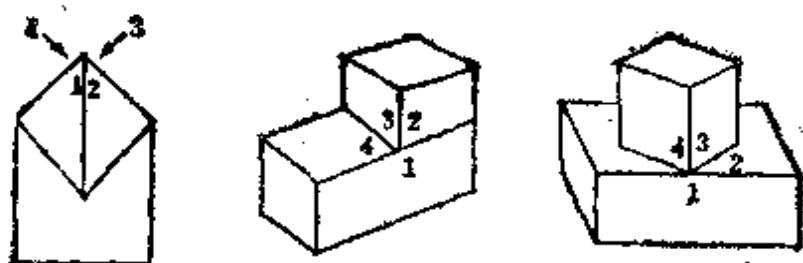


图 9.5 有四平面形成绩点的例子

的线条画不会因观察点稍有变动即引起结点类型的变化（即不取

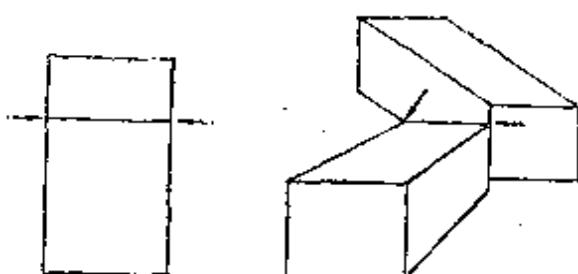


图 9.6 非一般观察点的例子

图 9.6 所示的情况，如左面的立方体线条画，观察点稍偏左移，就有好几个结点类型发生变化）。在这些假定下，结点的类型减少为只有 L型、箭型、叉型和 T型这 4 种，从而使构成这些结点的线条标记的组合数目达到实际上可能处理的程度。

由于每一根线条都可以有 4 种标记 (+、-、>、<)，因而这 4 类结点其线条可能标记的组合数为

L型: $4^2 = 16$ 种

箭型、叉型、T型: $4^3 = 64$ 种

整个线条画线条标记的组合数为 208 种。考虑到在三面顶点的假定下，有许多种标记组合现实世界是不可能实现的（如图 9.7 的标记组合，实际物体是不存在的），排除了这些不现实的组合后，实际上可能的标记组合只有：L型 6 种、叉型 5 种、T型 4 种、箭型 3 种。这 18 种标记组合如图 9.8 所示，

这就是利用实际物理世界的知识得到的结点约束的条件，我们只要用一种所谓的象限观察法就很容易得到图 9.8 所示的约束条件。

根据三平面顶点的假定，并利用三个平面顶点的三个平面把空间划分为八个象限，一个物体要形成一个顶点显然必须占据其中一个或几个象限的空间，因而可以先考虑该物体占据这 8 个象



图 9.7 不现实的结点线条标记

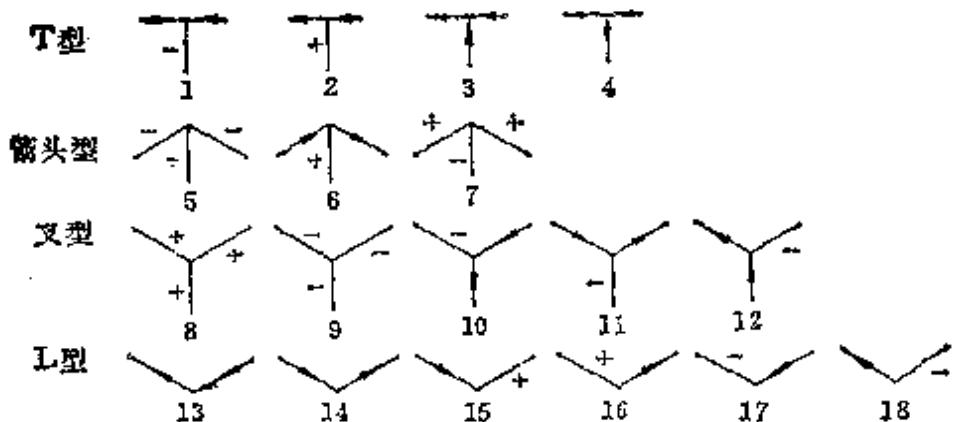


图 9.8 18种现实的结点线条标记

限所有可能的方式，再从未占据的象限来观察所形成的顶点，就可以给出所有可能的实体顶点的各种组合标记。下面就来说明这个过程。

我们先来看一下占据一个象限的立方体顶点 V ，从其他不同象限点观察时可能形成的结点标记。图9.9 (a) 表示出这个立方体顶点 V 有关的三个平面把空间分割成 8 个象限时所具有观察点 A、B、C、…G，以及从这些点观察 V 时所得到的结点标记，图 9.9 (b) 给出了 4 种可能的标记。同样的做法，可以得到占据 3 个象限时顶点 V 的各种标记如图 9.10 所示，共有 5 种不同的标记。图 9.11 给出占据 7 个象限时的 1 种标记。类似地可给出占据象限的其他情况，把所得到的结点类型汇总起来就是图 9.8 所给出的结果。

3. 约束条件的利用

利用图 9.8 给出的 18 种物理上可实现的结点类型作为约束条件，通过搜索并把约束条件加以传播，来对某一个线条画的所有线条选择合适的标记，这就是约束满足法的基本思想。也就是说约束满足法就是利用现实结点类型的知识，来引导搜索过程，

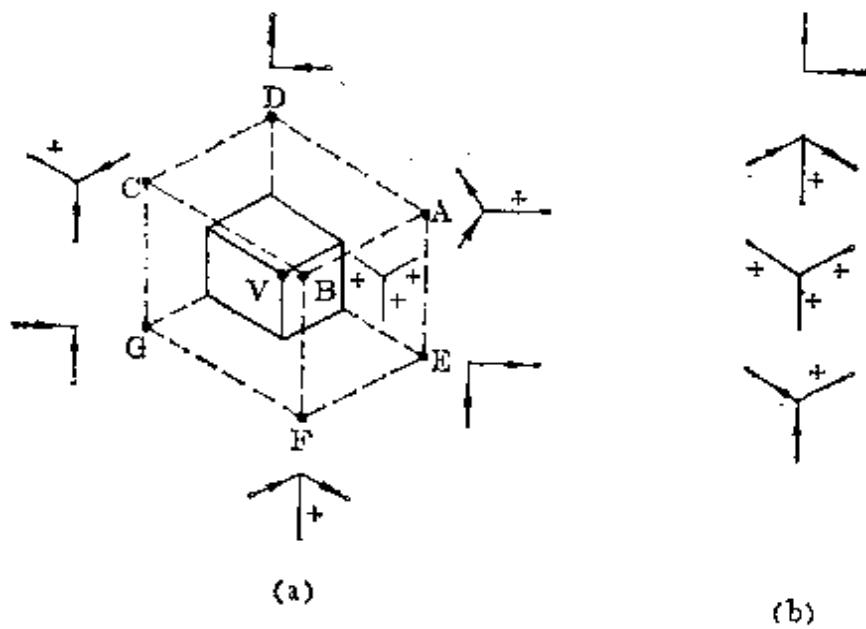


图 9.9 占1个象限立方体顶点的标记类型

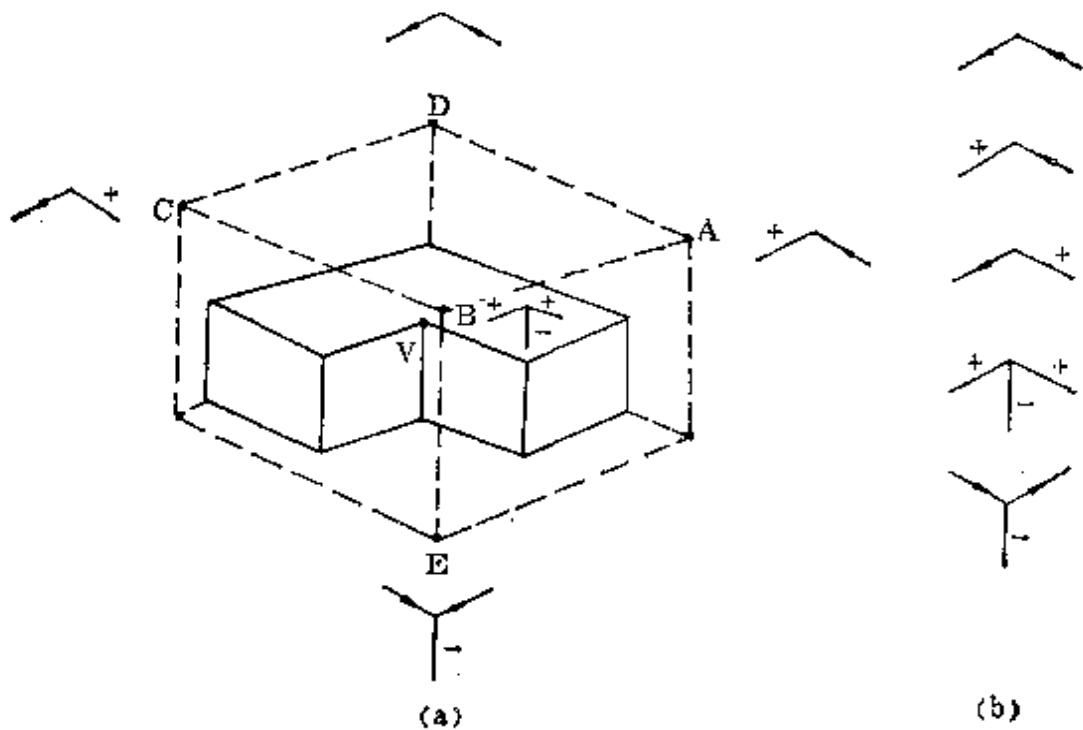


图 9.10 占3个象限立方体顶点的标记类型

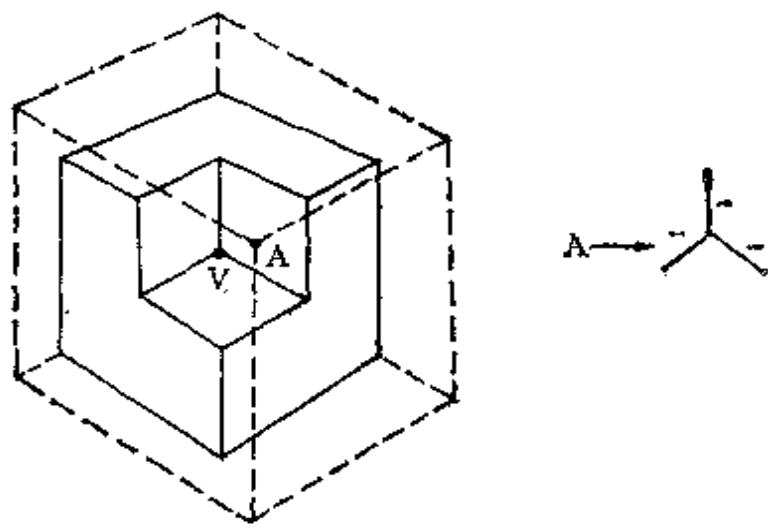


图 9.11 占7个象限立方体顶点的标记类型

约束条件一经确定，就立即加以应用，以限制线条标记数目在尽可能小的范围内进行选择，直至把全部的线条都标记完了。下面先通过几个简例来说明约束条件的利用，然后再介绍约束满足法的 Waltz 算法。

对图 9.12 (a) 的线条画，先标记其边界线如图 (b) 所示，接着按 A、B、…、F 标号顺序并根据约束条件来确定对应的结点类型，给出尚未标记线条的标记。设从结点 A 开始，A 是 L 型结点，它与第 13 种类型一致，接着考察结点 B，这个箭头结点由于箭头标记已确定，所以只有第 6 种标记一种选择。类似地根据约束的传播，D、F 也应按第 6 种来标记，最后得到如 (c) 的结果。最后来考察结点 G，这个叉型结点根据结点 B 的标记，BG 这条边必须是 + 标记，由此这个叉型结点只能是第 8 种选择，结果如图 (d) 所示。这就是利用顶点标记的约束条件，正确地识别了顶点 G 是由 3 条凸边组成。整个搜索过程如图 (e) 所示。

我们再来看一下图 9.13 的标记过程。设已知 A、B、C 三个结点有一个凹形线已标记，现在要对未标记的线条进行标记。假设

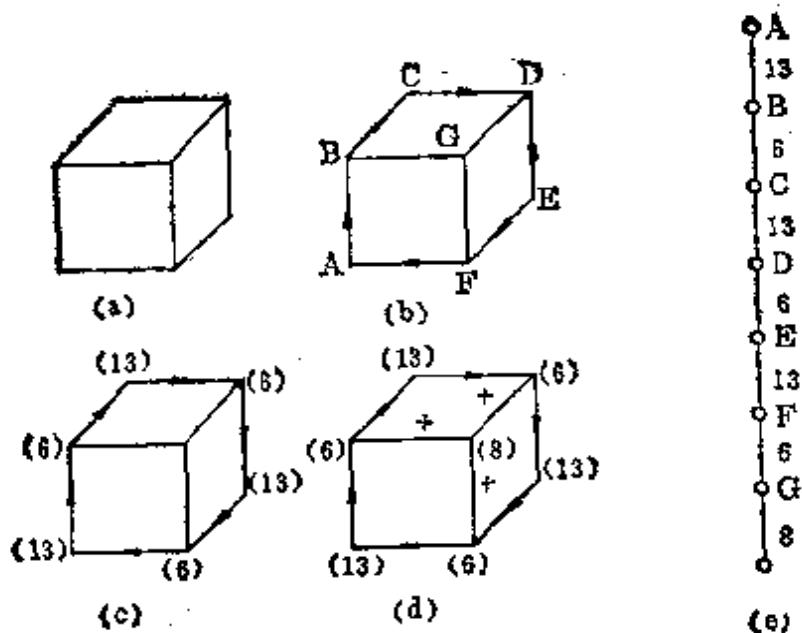


图 9.12 一个简例的标记过程

按 A、B、…、G 的顺序搜索，对 A、B、C 三个叉型结点，只有一条线有一标记，因此有第 9 和第 11 两种选择，接着考察 D，根据约束条件，只有两种组合 D 才可能进行标记。整个搜索过

程如图 9.14 所示，其中图 (b) 是按 A、D、B、E、C、F、G 的顺序进行搜索的结果。由此看出若能选择合适的顺序，能更有利地传播约束条件，使搜索的范围更早地得到限制。一般情况下，标记过程是首先标记边界线，然后从某一个约束最强的结点开始搜索，依次不断传播约束条件，直到所有线条标记完为止。

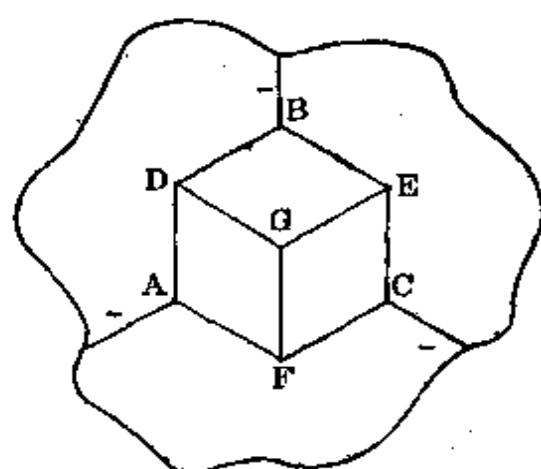


图 9.13 另一个要标记的线条画

Waltz 标记算法如下：

(1) 寻找景物的边界(边界线之外应不含有该物体的结点)，

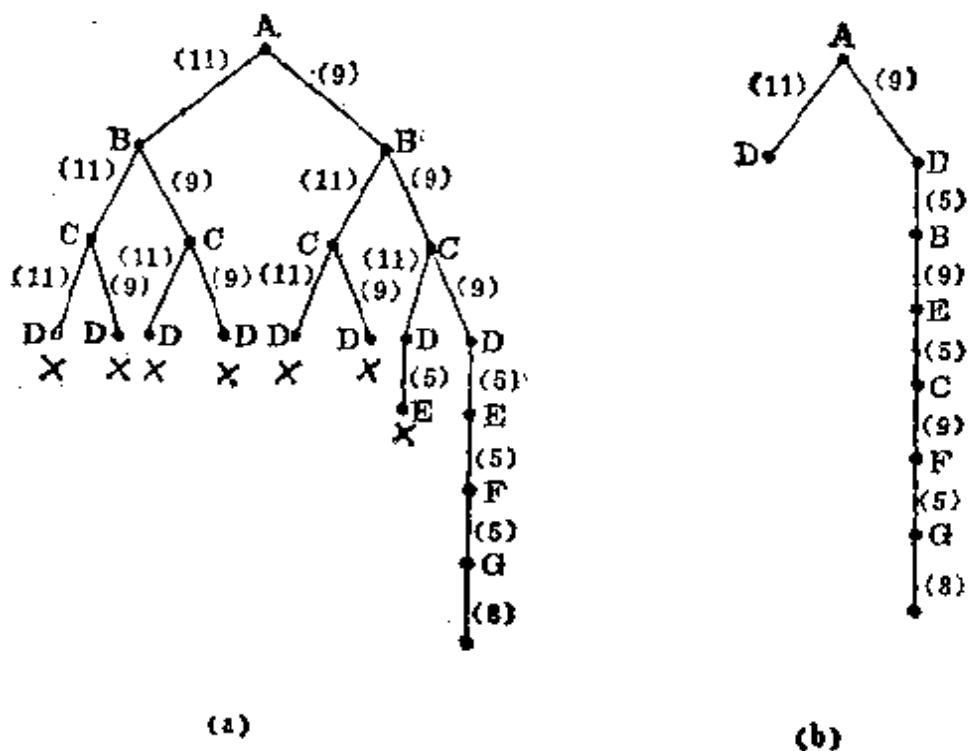


图 9.14 两种顺序的搜索过程

并按右侧法则进行标记。

(2) 按搜索顺序对线条画的顶点进行编号：

① 从边界线的任一顶点开始，因为边界线标记已知，由它们构成的顶点有较强的约束条件可以利用；

② 从初始顶点开始，沿其边界线邻接顶点的次序依次进行编号；

③ 从某一个已编号的顶点向未编号的邻接内部顶点依次进行编号，因为通常总是对已标记顶点相邻的内部顶点先进行标记，约束条件就可得到最大的利用。

(3) 按标号顺序考察每一个顶点 V ，并按以下步骤进行标记：

- ① 根据图9.8的标记集，对 V 给出一个可能的顶点标记表；
- ② 根据局部的约束条件，看是否可删除某些标记。为此应检

查每一个与 V 相邻并已访问过的顶点 A ，检查 V 的每一个选择标记，看一看是否有一种办法来标记 V 和 A 之间的线段，使得 A 的标记表至少有一个不与 V 发生矛盾的标记，然后从 V 的标记表中删去有矛盾的任何标记；

③ 利用附加在 V 上的标记集来约束与 V 相邻顶点的标记。对上一步所访问过的每一顶点 A ，执行以下各步：

- 1) 删去 A 中至少与 V 的一个标记不相容的所有标记；
- 2) 若任何不相容的标记都已删去，则通过考察邻接于 A 的各个顶点并检查与当前附加于 A 的有限制的标记集的一致性，继续传播约束；
- 3) 继续传播约束直到再没有要标记的邻接顶点为止，或者直到现存的标记集不再发生变化为止。

4. 算法的推广应用问题

前面讨论的结果都是在无阴影、无缝隙线和只有三平面结点的线条画这几个假定下进行的。对于比较简单的图形，只要线条存在可实现的标记，这个算法就可以找到这个唯一、正确的图形标记，如果图形有二义性，则该算法结束时，至少会有一个顶点会给出多于一种的标记法。实际上 Waltz 算法已推广应用到较复杂的图形，这种图形可包含阴影线、缝隙线和四平面的结点，在这种情况下，线条的标记种类将增加到 11 种，结点的类型可达 10 种。由于线条种类增多，每一个结点标记的组合数（无约束）大为增加，当然物理上可实现的标记数（有约束）也相应增加，尽管如此，但对每种结点有约束标记数与无约束标记数之比值却比简单图形的情况减少很多。可见引进阴影信息，加强了约束条件，有利于提高搜索效率。关于这个问题更全面的讨论，可参阅有关专著。

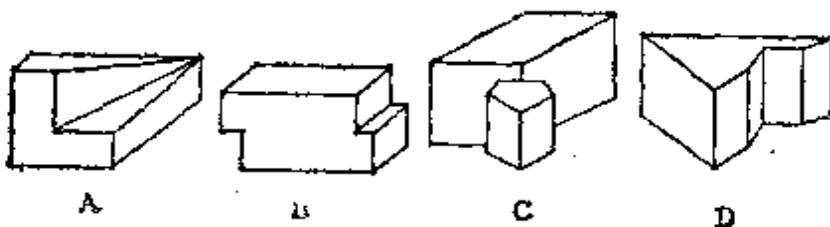
9.4 小结

1. 感知是研究外部世界各种信息描述和理解问题，在人工智能中主要是研究景物和语音的识别和理解问题。这是一个比较困难的问题，目前虽然已经提出一些方法和系统，但仍面临许多困难，还在开展艰巨的研究工作。

2. 解决整个感知问题涉及信号处理和人工智能的有关技术。本章只是简单积木世界理解问题，并通过约束满足法的求解过程来讨论人工智能技术的应用。

习题

9.1 指出下列图形哪些是三平面顶点的图形。



9.2 用 Waltz 算法标记如下的三平面顶点图形。



9.3 能否给出一个有二义性理解的图例来说明Waltz 算法不能找到唯一的标记。

第十章 学习

10.1 概述

学习是人类特有的一种能力，自从人工智能学科诞生以来，研究者们就开始研究机器学习问题，至今这一课题仍然是人工智能研究中最富有挑战性和令人向往的长期奋斗目标之一。有些人认为如果机器只能做人们告诉它要做的事，不能学着去做新的事情和适应新的环境，那么机器就不能认为是有智能，可见这种看法十分强调机器的学习能力。当前机器学习的主要课题是研究学习过程和建造计算机学习模型，以便实现机器智能。

关于学习的定义有各种说法。心理学家认为：学习是系统积累经验以改善其性能的过程，学习是个体经验的获得以及行为变化的过程。人工智能研究者则提出：学习是获取新的陈述知识，是通过指教和实践活动去获取技巧，是把新知识组织成为较一般更有效的各种表示，是通过观察和试验去发现新事实和新理论。显然第一种说法较为一般，不便于使用，后一种说法比较具体，可以利用人工智能的技术加以实现。可以看出学习系统的研究与概念形成、归纳推理、假设猜想和科学发现等课题的研究紧密相关。

在人工智能领域中研究学习问题经历了三个阶段。第一阶段是50年代末开始的“乐观”时期，这一时期的主要工作集中在能修改系统本身以适应环境变化的自组织系统方面。另外还从仿生学角度进行神经元的各种模拟，企图从生理结构上模仿大脑与神经系统的学习机理，但由于这些研究受到理论上的一些限制，都未

能建造出较复杂的或有智能的学习系统来。第二阶段始于70年代初，这一时期采取的基本观点是认为学习是一个复杂和艰难的过程，因而不能指望学习系统在没有任何背景知识的基础上，能学到高级的概念。于是出现了一方面深入研究简单的学习问题（如学习单一概念），另一方面则把大量领域专业知识纳入学习系统的局面。第三阶段受到专家系统的发展，需要解决知识获取问题的推动。这一时期研究各种形式的学习系统，不仅探讨前两阶段所进行的机械学习和根据例子学习的问题，还研究指点学习和类比学习。

当前机器学习围绕三个主要研究方向进行：

（1）面向任务：在预定的一些任务中，分析和开发学习系统，以便改善完成任务的水平，这是专家系统研究中提出的研究问题。

（2）认知模拟：主要研究人类学习过程及其计算机的行为模拟，这是从心理学角度研究的问题。

（3）理论分析研究：从理论上探讨各种可能学习方法的空间和独立于应用领域之外的各种算法。

这三个研究方向各有自己的研究目标，每一个方向的进展都会促进另一个方向的研究。例如要研究可能学习方法的空间这个理论问题，可先从人类学习过程开始，因为已经知道人类的一些固定的学习行为。同样地人类学习的心理学研究可能有助于理论的分析，并导致提出各种可能的学习模型。有些面向任务的研究中，需要获取特殊形式的知识，这又会提出新的理论问题需要解决。总之专家系统、认知模拟和理论探讨三方面的研究都将促进各方面问题和学习基本概念的交叉结合，推动了整个机器学习的研究工作。

10.2 机器学习的分类

关于机器学习的分类问题，可以从不同的角度来划分，比较有实际意义的是根据所使用的基本学习策略、根据知识的表示或获取的技巧和根据实现系统的应用领域这三种分类法，下面作一简单介绍。

1. 根据基本学习策略的分类法

学习策略是按照学习程序对提供的信息所做的推理数量有多少来区分的。一种极端情况是没有任何推理功能，要增加系统知识时就得靠外界环境作大量的输入新知识的工作；另一种极端情况则是学习程序有相当数量的推理，可以根据试验和观察推导出有组织的知识，这样系统就可能独立地发现新理论或发明新概念。介于这两者之间就是学习程序有一定的推理能力，能适应减轻外界环境负担的各种折衷情况。下面的分类就是按照这种思想进行的。

（1）死记学习：学习程序不具有推理能力，靠直接向系统植入知识的办法增加新知识，例如通过外界的力量构造或修改程序系统，或者通过记住输入信息中给定的事实和数据等方法来获取知识。

（2）指点（教授）学习：通过教师或其他知识系统（如教科书）来获取知识，学习程序要把输入语言携带的知识转化为内部可用的一种表示，并要以有效使用知识的优先级顺序将新知识组织起来，这样学习程序有一定的推理能力，但大部分的组织知识的工作仍然由外界环境负担。

（3）类比学习：通过相似性对比的办法来获取新事实或新技巧。学习程序把和新概念或新技巧相似的现有知识转化并扩充

为有效适用于新情况的某种形式，这要求比前两种学习有较强的推理能力，类似于有关问题的事实或技巧必须从内存中检索出来，然后把检索的知识进行转换并应用于新情况，得到的结果还要存储起来供以后使用。

（4）概念学习：这是归纳学习的一个特例，学习程序根据给定某一概念的一组正例和反例，归纳出一个一般化的概念描述，它可描述所有的正例而无一反例。这要求更强的推理能力，无须由外界提供一般的概念，也不必提供相似的概念作为学习新概念的启发。

（5）根据观察和发现的学习：这是归纳学习的最一般形式，也称为无指导的学习。这种学习包括发现系统、理论形成的任务、创立分类判据来建造分类的层次体系，以及无须从外界教师获益的类似任务等。学习系统具有高级推理的能力，外界不提供任何信息，靠系统的观察和发现来获取新知识。

2. 根据获取知识类型的分类法

学习系统可以获取行为的规则、物理对象的描述、求解问题的经验知识、对某个取样空间的分类法以及完成各种任务中有用的许多其他各类知识。下面按获取知识类型表示形式列出。

（1）代数表达式的各种参数：学习包括调整某一固定函数形的代数表达式中各种数值参数或系数，以便达到所希望的性能。

（2）决策树：一些学习系统获取决策树来区分物体之间的分类，决策树的节点对应于选择对象的属性，边对应于属性的预先确定可供选择的值，叶节点对应于某种识别分类的物体。

（3）形式文法：在识别某种特定语言的学习中，要从该语言序列表达式中归纳出形式文法。通常这些文件被表示为正则表达式、有限自动机、上下文无关文法规则或转换规则等。

(4) 产生式规则：产生式规则是学习系统中广泛使用的知识表示形式，有4种基本运算可用来获取和改进产生式规则：创立（建立新规则）、一般化（削弱条件部分的限制，展宽规则的应用范围）、特殊化（增强条件部分的约束，限制规则只应用于个别场合）、组合化（把多条规则组合为一条较大的规则，以利于消除冗余条件或冗余的行动）。

(5) 基于形式化的逻辑表达式和相关的形式体系：这些通用的表示法已用于系统输入个体对象的形式化描述和系统输出生成概念的形式化描述。其形式是形式化的逻辑表达式，包括有命题、谓词、有限值变量、变量范围有限制的语句等。

(6) 图或网络：图和网络在许多领域中是一种较方便和有效的表示法，有几种学习技术利用了图匹配和图传递的模式来比较和标注知识。

(7) 框架和图式：当系统获取综合性的计划时，必须能够将计划当作为单元整体加以表示和进行处理，框架和图式提供了这种较大单元的表示法。此外在归纳和改进各种行为规则中，必须记录和比较其他一些信息，图式表示法是一种合适的形式体系。

(8) 计算机程序和其他过程式编码：有些学习系统的目是获取某种能力，以便高效率地执行某一个特殊的过程，而不是对该过程内部结构的推理。大多数的自动程序设计系统都属这一类，除了计算机程序外，过程编码涉及人类运动的技巧，机器人操作器的指令序列以及其他机器的技巧。

(9) 分类学：根据观察的学习可以把全局结构化的领域对象归纳为分类体系，把对象描述分成群，组成新的分类，并形成层次分类要求系统建立有关的分类判据。

(10) 多重表示：有些知识获取系统对新获取的知识使用几种表示格式，例如有些发现和理论形式化系统，它们获取概念、

(4) 产生式规则：产生式规则是学习系统中广泛使用的知识表示形式，有4种基本运算可用来获取和改进产生式规则：创立（建立新规则）、一般化（削弱条件部分的限制，展宽规则的应用范围）、特殊化（增强条件部分的约束，限制规则只应用于个别场合）、组合化（把多条规则组合为一条较大的规则，以利于消除冗余条件或冗余的行动）。

(5) 基于形式化的逻辑表达式和相关的形式体系：这些通用的表示法已用于系统输入个体对象的形式化描述和系统输出生成概念的形式化描述。其形式是形式化的逻辑表达式，包括有命题、谓词、有限值变量、变量范围有限制的语句等。

(6) 图或网络：图和网络在许多领域中是一种较方便和有效的表示法，有几种学习技术利用了图匹配和图传递的模式来比较和标注知识。

(7) 框架和图式：当系统获取综合性的计划时，必须能够将计划当作为单元整体加以表示和进行处理，框架和图式提供了这种较大单元的表示法。此外在归纳和改进各种行为规则中，必须记录和比较其他一些信息，图式表示法是一种合适的形式体系。

(8) 计算机程序和其他过程式编码：有些学习系统的目标是获取某种能力，以便高效率地执行某一个特殊的过程，而不是对该过程内部结构的推理。大多数的自动程序设计系统都属这一类，除了计算机程序外，过程编码涉及人类运动的技巧，机器人操作器的指令序列以及其他机器的技巧。

(9) 分类学：根据观察的学习可以把全局结构化的领域对象归纳为分类体系，把对象描述分成群，组成新的分类，并形成层次分类要求系统建立有关的分类判据。

(10) 多重表示：有些知识获取系统对新获取的知识使用几种表示格式，例如有些发现和理论形式化系统，它们获取概念、

(4) 产生式规则：产生式规则是学习系统中广泛使用的知识表示形式，有4种基本运算可用来获取和改进产生式规则：创立（建立新规则）、一般化（削弱条件部分的限制，展宽规则的应用范围）、特殊化（增强条件部分的约束，限制规则只应用于个别场合）、组合化（把多条规则组合为一条较大的规则，以利于消除冗余条件或冗余的行动）。

(5) 基于形式化的逻辑表达式和相关的形式体系：这些通用的表示法已用于系统输入个体对象的形式化描述和系统输出生成概念的形式化描述。其形式是形式化的逻辑表达式，包括有命题、谓词、有限值变量、变量范围有限制的语句等。

(6) 图或网络：图和网络在许多领域中是一种较方便和有效的表示法，有几种学习技术利用了图匹配和图传递的模式来比较和标注知识。

(7) 框架和图式：当系统获取综合性的计划时，必须能够将计划当作为单元整体加以表示和进行处理，框架和图式提供了这种较大单元的表示法。此外在归纳和改进各种行为规则中，必须记录和比较其他一些信息，图式表示法是一种合适的形式体系。

(8) 计算机程序和其他过程式编码：有些学习系统的目是获取某种能力，以便高效率地执行某一个特殊的过程，而不是对该过程内部结构的推理。大多数的自动程序设计系统都属这一类，除了计算机程序外，过程编码涉及人类运动的技巧，机器人操作器的指令序列以及其他机器的技巧。

(9) 分类学：根据观察的学习可以把全局结构化的领域对象归纳为分类体系，把对象描述分成群，组成新的分类，并形成层次分类要求系统建立有关的分类判据。

(10) 多重表示：有些知识获取系统对新获取的知识使用几种表示格式，例如有些发现和理论形式化系统，它们获取概念、

(4) 产生式规则：产生式规则是学习系统中广泛使用的知识表示形式，有4种基本运算可用来获取和改进产生式规则：创立（建立新规则）、一般化（削弱条件部分的限制，展宽规则的应用范围）、特殊化（增强条件部分的约束，限制规则只应用于个别场合）、组合化（把多条规则组合为一条较大的规则，以利于消除冗余条件或冗余的行动）。

(5) 基于形式化的逻辑表达式和相关的形式体系：这些通用的表示法已用于系统输入个体对象的形式化描述和系统输出生成概念的形式化描述。其形式是形式化的逻辑表达式，包括有命题、谓词、有限值变量、变量范围有限制的语句等。

(6) 图或网络：图和网络在许多领域中是一种较方便和有效的表示法，有几种学习技术利用了图匹配和图传递的模式来比较和标注知识。

(7) 框架和图式：当系统获取综合性的计划时，必须能够将计划当作为单元整体加以表示和进行处理，框架和图式提供了这种较大单元的表示法。此外在归纳和改进各种行为规则中，必须记录和比较其他一些信息，图式表示法是一种合适的形式体系。

(8) 计算机程序和其他过程式编码：有些学习系统的目是获取某种能力，以便高效率地执行某一个特殊的过程，而不是对该过程内部结构的推理。大多数的自动程序设计系统都属这一类，除了计算机程序外，过程编码涉及人类运动的技巧，机器人操作器的指令序列以及其他机器的技巧。

(9) 分类学：根据观察的学习可以把全局结构化的领域对象归纳为分类体系，把对象描述分成群，组成新的分类，并形成层次分类要求系统建立有关的分类判据。

(10) 多重表示：有些知识获取系统对新获取的知识使用几种表示格式，例如有些发现和理论形式化系统，它们获取概念、

(4) 产生式规则：产生式规则是学习系统中广泛使用的知识表示形式，有4种基本运算可用来获取和改进产生式规则：创立（建立新规则）、一般化（削弱条件部分的限制，展宽规则的应用范围）、特殊化（增强条件部分的约束，限制规则只应用于个别场合）、组合化（把多条规则组合为一条较大的规则，以利于消除冗余条件或冗余的行动）。

(5) 基于形式化的逻辑表达式和相关的形式体系：这些通用的表示法已用于系统输入个体对象的形式化描述和系统输出生成概念的形式化描述。其形式是形式化的逻辑表达式，包括有命题、谓词、有限值变量、变量范围有限制的语句等。

(6) 图或网络：图和网络在许多领域中是一种较方便和有效的表示法，有几种学习技术利用了图匹配和图传递的模式来比较和标注知识。

(7) 框架和图式：当系统获取综合性的计划时，必须能够将计划当作为单元整体加以表示和进行处理，框架和图式提供了这种较大单元的表示法。此外在归纳和改进各种行为规则中，必须记录和比较其他一些信息，图式表示法是一种合适的形式体系。

(8) 计算机程序和其他过程式编码：有些学习系统的目是获取某种能力，以便高效率地执行某一个特殊的过程，而不是对该过程内部结构的推理。大多数的自动程序设计系统都属这一类，除了计算机程序外，过程编码涉及人类运动的技巧，机器人操作器的指令序列以及其他机器的技巧。

(9) 分类学：根据观察的学习可以把全局结构化的领域对象归纳为分类体系，把对象描述分成群，组成新的分类，并形成层次分类要求系统建立有关的分类判据。

(10) 多重表示：有些知识获取系统对新获取的知识使用几种表示格式，例如有些发现和理论形式化系统，它们获取概念、

(4) 产生式规则：产生式规则是学习系统中广泛使用的知识表示形式，有4种基本运算可用来获取和改进产生式规则：创立（建立新规则）、一般化（削弱条件部分的限制，展宽规则的应用范围）、特殊化（增强条件部分的约束，限制规则只应用于个别场合）、组合化（把多条规则组合为一条较大的规则，以利于消除冗余条件或冗余的行动）。

(5) 基于形式化的逻辑表达式和相关的形式体系：这些通用的表示法已用于系统输入个体对象的形式化描述和系统输出生成概念的形式化描述。其形式是形式化的逻辑表达式，包括有命题、谓词、有限值变量、变量范围有限制的语句等。

(6) 图或网络：图和网络在许多领域中是一种较方便和有效的表示法，有几种学习技术利用了图匹配和图传递的模式来比较和标注知识。

(7) 框架和图式：当系统获取综合性的计划时，必须能够将计划当作为单元整体加以表示和进行处理，框架和图式提供了这种较大单元的表示法。此外在归纳和改进各种行为规则中，必须记录和比较其他一些信息，图式表示法是一种合适的形式体系。

(8) 计算机程序和其他过程式编码：有些学习系统的目是获取某种能力，以便高效率地执行某一个特殊的过程，而不是对该过程内部结构的推理。大多数的自动程序设计系统都属这一类，除了计算机程序外，过程编码涉及人类运动的技巧，机器人操作器的指令序列以及其他机器的技巧。

(9) 分类学：根据观察的学习可以把全局结构化的领域对象归纳为分类体系，把对象描述分成群，组成新的分类，并形成层次分类要求系统建立有关的分类判据。

(10) 多重表示：有些知识获取系统对新获取的知识使用几种表示格式，例如有些发现和理论形式化系统，它们获取概念、

(4) 产生式规则：产生式规则是学习系统中广泛使用的知识表示形式，有4种基本运算可用来获取和改进产生式规则：创立（建立新规则）、一般化（削弱条件部分的限制，展宽规则的应用范围）、特殊化（增强条件部分的约束，限制规则只应用于个别场合）、组合化（把多条规则组合为一条较大的规则，以利于消除冗余条件或冗余的行动）。

(5) 基于形式化的逻辑表达式和相关的形式体系：这些通用的表示法已用于系统输入个体对象的形式化描述和系统输出生成概念的形式化描述。其形式是形式化的逻辑表达式，包括有命题、谓词、有限值变量、变量范围有限制的语句等。

(6) 图或网络：图和网络在许多领域中是一种较方便和有效的表示法，有几种学习技术利用了图匹配和图传递的模式来比较和标注知识。

(7) 框架和图式：当系统获取综合性的计划时，必须能够将计划当作为单元整体加以表示和进行处理，框架和图式提供了这种较大单元的表示法。此外在归纳和改进各种行为规则中，必须记录和比较其他一些信息，图式表示法是一种合适的形式体系。

(8) 计算机程序和其他过程式编码：有些学习系统的目是获取某种能力，以便高效率地执行某一个特殊的过程，而不是对该过程内部结构的推理。大多数的自动程序设计系统都属这一类，除了计算机程序外，过程编码涉及人类运动的技巧，机器人操作器的指令序列以及其他机器的技巧。

(9) 分类学：根据观察的学习可以把全局结构化的领域对象归纳为分类体系，把对象描述分成群，组成新的分类，并形成层次分类要求系统建立有关的分类判据。

(10) 多重表示：有些知识获取系统对新获取的知识使用几种表示格式，例如有些发现和理论形式化系统，它们获取概念、

(4) 产生式规则：产生式规则是学习系统中广泛使用的知识表示形式，有4种基本运算可用来获取和改进产生式规则：创立（建立新规则）、一般化（削弱条件部分的限制，展宽规则的应用范围）、特殊化（增强条件部分的约束，限制规则只应用于个别场合）、组合化（把多条规则组合为一条较大的规则，以利于消除冗余条件或冗余的行动）。

(5) 基于形式化的逻辑表达式和相关的形式体系：这些通用的表示法已用于系统输入个体对象的形式化描述和系统输出生成概念的形式化描述。其形式是形式化的逻辑表达式，包括有命题、谓词、有限值变量、变量范围有限制的语句等。

(6) 图或网络：图和网络在许多领域中是一种较方便和有效的表示法，有几种学习技术利用了图匹配和图传递的模式来比较和标注知识。

(7) 框架和图式：当系统获取综合性的计划时，必须能够将计划当作为单元整体加以表示和进行处理，框架和图式提供了这种较大单元的表示法。此外在归纳和改进各种行为规则中，必须记录和比较其他一些信息，图式表示法是一种合适的形式体系。

(8) 计算机程序和其他过程式编码：有些学习系统的目是获取某种能力，以便高效率地执行某一个特殊的过程，而不是对该过程内部结构的推理。大多数的自动程序设计系统都属这一类，除了计算机程序外，过程编码涉及人类运动的技巧，机器人操作器的指令序列以及其他机器的技巧。

(9) 分类学：根据观察的学习可以把全局结构化的领域对象归纳为分类体系，把对象描述分成群，组成新的分类，并形成层次分类要求系统建立有关的分类判据。

(10) 多重表示：有些知识获取系统对新获取的知识使用几种表示格式，例如有些发现和理论形式化系统，它们获取概念、

(4) 产生式规则：产生式规则是学习系统中广泛使用的知识表示形式，有4种基本运算可用来获取和改进产生式规则：创立（建立新规则）、一般化（削弱条件部分的限制，展宽规则的应用范围）、特殊化（增强条件部分的约束，限制规则只应用于个别场合）、组合化（把多条规则组合为一条较大的规则，以利于消除冗余条件或冗余的行动）。

(5) 基于形式化的逻辑表达式和相关的形式体系：这些通用的表示法已用于系统输入个体对象的形式化描述和系统输出生成概念的形式化描述。其形式是形式化的逻辑表达式，包括有命题、谓词、有限值变量、变量范围有限制的语句等。

(6) 图或网络：图和网络在许多领域中是一种较方便和有效的表示法，有几种学习技术利用了图匹配和图传递的模式来比较和标注知识。

(7) 框架和图式：当系统获取综合性的计划时，必须能够将计划当作为单元整体加以表示和进行处理，框架和图式提供了这种较大单元的表示法。此外在归纳和改进各种行为规则中，必须记录和比较其他一些信息，图式表示法是一种合适的形式体系。

(8) 计算机程序和其他过程式编码：有些学习系统的目是获取某种能力，以便高效率地执行某一个特殊的过程，而不是对该过程内部结构的推理。大多数的自动程序设计系统都属这一类，除了计算机程序外，过程编码涉及人类运动的技巧，机器人操作器的指令序列以及其他机器的技巧。

(9) 分类学：根据观察的学习可以把全局结构化的领域对象归纳为分类体系，把对象描述分成群，组成新的分类，并形成层次分类要求系统建立有关的分类判据。

(10) 多重表示：有些知识获取系统对新获取的知识使用几种表示格式，例如有些发现和理论形式化系统，它们获取概念、

(4) 产生式规则：产生式规则是学习系统中广泛使用的知识表示形式，有4种基本运算可用来获取和改进产生式规则：创立（建立新规则）、一般化（削弱条件部分的限制，展宽规则的应用范围）、特殊化（增强条件部分的约束，限制规则只应用于个别场合）、组合化（把多条规则组合为一条较大的规则，以利于消除冗余条件或冗余的行动）。

(5) 基于形式化的逻辑表达式和相关的形式体系：这些通用的表示法已用于系统输入个体对象的形式化描述和系统输出生成概念的形式化描述。其形式是形式化的逻辑表达式，包括有命题、谓词、有限值变量、变量范围有限制的语句等。

(6) 图或网络：图和网络在许多领域中是一种较方便和有效的表示法，有几种学习技术利用了图匹配和图传递的模式来比较和标注知识。

(7) 框架和图式：当系统获取综合性的计划时，必须能够将计划当作为单元整体加以表示和进行处理，框架和图式提供了这种较大单元的表示法。此外在归纳和改进各种行为规则中，必须记录和比较其他一些信息，图式表示法是一种合适的形式体系。

(8) 计算机程序和其他过程式编码：有些学习系统的目是获取某种能力，以便高效率地执行某一个特殊的过程，而不是对该过程内部结构的推理。大多数的自动程序设计系统都属这一类，除了计算机程序外，过程编码涉及人类运动的技巧，机器人操作器的指令序列以及其他机器的技巧。

(9) 分类学：根据观察的学习可以把全局结构化的领域对象归纳为分类体系，把对象描述分成群，组成新的分类，并形成层次分类要求系统建立有关的分类判据。

(10) 多重表示：有些知识获取系统对新获取的知识使用几种表示格式，例如有些发现和理论形式化系统，它们获取概念、

(4) 产生式规则：产生式规则是学习系统中广泛使用的知识表示形式，有4种基本运算可用来获取和改进产生式规则：创立（建立新规则）、一般化（削弱条件部分的限制，展宽规则的应用范围）、特殊化（增强条件部分的约束，限制规则只应用于个别场合）、组合化（把多条规则组合为一条较大的规则，以利于消除冗余条件或冗余的行动）。

(5) 基于形式化的逻辑表达式和相关的形式体系：这些通用的表示法已用于系统输入个体对象的形式化描述和系统输出生成概念的形式化描述。其形式是形式化的逻辑表达式，包括有命题、谓词、有限值变量、变量范围有限制的语句等。

(6) 图或网络：图和网络在许多领域中是一种较方便和有效的表示法，有几种学习技术利用了图匹配和图传递的模式来比较和标注知识。

(7) 框架和图式：当系统获取综合性的计划时，必须能够将计划当作为单元整体加以表示和进行处理，框架和图式提供了这种较大单元的表示法。此外在归纳和改进各种行为规则中，必须记录和比较其他一些信息，图式表示法是一种合适的形式体系。

(8) 计算机程序和其他过程式编码：有些学习系统的目是获取某种能力，以便高效率地执行某一个特殊的过程，而不是对该过程内部结构的推理。大多数的自动程序设计系统都属这一类，除了计算机程序外，过程编码涉及人类运动的技巧，机器人操作器的指令序列以及其他机器的技巧。

(9) 分类学：根据观察的学习可以把全局结构化的领域对象归纳为分类体系，把对象描述分成群，组成新的分类，并形成层次分类要求系统建立有关的分类判据。

(10) 多重表示：有些知识获取系统对新获取的知识使用几种表示格式，例如有些发现和理论形式化系统，它们获取概念、

(4) 产生式规则：产生式规则是学习系统中广泛使用的知识表示形式，有4种基本运算可用来获取和改进产生式规则：创立（建立新规则）、一般化（削弱条件部分的限制，展宽规则的应用范围）、特殊化（增强条件部分的约束，限制规则只应用于个别场合）、组合化（把多条规则组合为一条较大的规则，以利于消除冗余条件或冗余的行动）。

(5) 基于形式化的逻辑表达式和相关的形式体系：这些通用的表示法已用于系统输入个体对象的形式化描述和系统输出生成概念的形式化描述。其形式是形式化的逻辑表达式，包括有命题、谓词、有限值变量、变量范围有限制的语句等。

(6) 图或网络：图和网络在许多领域中是一种较方便和有效的表示法，有几种学习技术利用了图匹配和图传递的模式来比较和标注知识。

(7) 框架和图式：当系统获取综合性的计划时，必须能够将计划当作为单元整体加以表示和进行处理，框架和图式提供了这种较大单元的表示法。此外在归纳和改进各种行为规则中，必须记录和比较其他一些信息，图式表示法是一种合适的形式体系。

(8) 计算机程序和其他过程式编码：有些学习系统的目是获取某种能力，以便高效率地执行某一个特殊的过程，而不是对该过程内部结构的推理。大多数的自动程序设计系统都属这一类，除了计算机程序外，过程编码涉及人类运动的技巧，机器人操作器的指令序列以及其他机器的技巧。

(9) 分类学：根据观察的学习可以把全局结构化的领域对象归纳为分类体系，把对象描述分成群，组成新的分类，并形成层次分类要求系统建立有关的分类判据。

(10) 多重表示：有些知识获取系统对新获取的知识使用几种表示格式，例如有些发现和理论形式化系统，它们获取概念、

(4) 产生式规则：产生式规则是学习系统中广泛使用的知识表示形式，有4种基本运算可用来获取和改进产生式规则：创立（建立新规则）、一般化（削弱条件部分的限制，展宽规则的应用范围）、特殊化（增强条件部分的约束，限制规则只应用于个别场合）、组合化（把多条规则组合为一条较大的规则，以利于消除冗余条件或冗余的行动）。

(5) 基于形式化的逻辑表达式和相关的形式体系：这些通用的表示法已用于系统输入个体对象的形式化描述和系统输出生成概念的形式化描述。其形式是形式化的逻辑表达式，包括有命题、谓词、有限值变量、变量范围有限制的语句等。

(6) 图或网络：图和网络在许多领域中是一种较方便和有效的表示法，有几种学习技术利用了图匹配和图传递的模式来比较和标注知识。

(7) 框架和图式：当系统获取综合性的计划时，必须能够将计划当作为单元整体加以表示和进行处理，框架和图式提供了这种较大单元的表示法。此外在归纳和改进各种行为规则中，必须记录和比较其他一些信息，图式表示法是一种合适的形式体系。

(8) 计算机程序和其他过程式编码：有些学习系统的目是获取某种能力，以便高效率地执行某一个特殊的过程，而不是对该过程内部结构的推理。大多数的自动程序设计系统都属这一类，除了计算机程序外，过程编码涉及人类运动的技巧，机器人操作器的指令序列以及其他机器的技巧。

(9) 分类学：根据观察的学习可以把全局结构化的领域对象归纳为分类体系，把对象描述分成群，组成新的分类，并形成层次分类要求系统建立有关的分类判据。

(10) 多重表示：有些知识获取系统对新获取的知识使用几种表示格式，例如有些发现和理论形式化系统，它们获取概念、

(4) 产生式规则：产生式规则是学习系统中广泛使用的知识表示形式，有4种基本运算可用来获取和改进产生式规则：创立（建立新规则）、一般化（削弱条件部分的限制，展宽规则的应用范围）、特殊化（增强条件部分的约束，限制规则只应用于个别场合）、组合化（把多条规则组合为一条较大的规则，以利于消除冗余条件或冗余的行动）。

(5) 基于形式化的逻辑表达式和相关的形式体系：这些通用的表示法已用于系统输入个体对象的形式化描述和系统输出生成概念的形式化描述。其形式是形式化的逻辑表达式，包括有命题、谓词、有限值变量、变量范围有限制的语句等。

(6) 图或网络：图和网络在许多领域中是一种较方便和有效的表示法，有几种学习技术利用了图匹配和图传递的模式来比较和标注知识。

(7) 框架和图式：当系统获取综合性的计划时，必须能够将计划当作为单元整体加以表示和进行处理，框架和图式提供了这种较大单元的表示法。此外在归纳和改进各种行为规则中，必须记录和比较其他一些信息，图式表示法是一种合适的形式体系。

(8) 计算机程序和其他过程式编码：有些学习系统的目是获取某种能力，以便高效率地执行某一个特殊的过程，而不是对该过程内部结构的推理。大多数的自动程序设计系统都属这一类，除了计算机程序外，过程编码涉及人类运动的技巧，机器人操作器的指令序列以及其他机器的技巧。

(9) 分类学：根据观察的学习可以把全局结构化的领域对象归纳为分类体系，把对象描述分成群，组成新的分类，并形成层次分类要求系统建立有关的分类判据。

(10) 多重表示：有些知识获取系统对新获取的知识使用几种表示格式，例如有些发现和理论形式化系统，它们获取概念、

(4) 产生式规则：产生式规则是学习系统中广泛使用的知识表示形式，有4种基本运算可用来获取和改进产生式规则：创立（建立新规则）、一般化（削弱条件部分的限制，展宽规则的应用范围）、特殊化（增强条件部分的约束，限制规则只应用于个别场合）、组合化（把多条规则组合为一条较大的规则，以利于消除冗余条件或冗余的行动）。

(5) 基于形式化的逻辑表达式和相关的形式体系：这些通用的表示法已用于系统输入个体对象的形式化描述和系统输出生成概念的形式化描述。其形式是形式化的逻辑表达式，包括有命题、谓词、有限值变量、变量范围有限制的语句等。

(6) 图或网络：图和网络在许多领域中是一种较方便和有效的表示法，有几种学习技术利用了图匹配和图传递的模式来比较和标注知识。

(7) 框架和图式：当系统获取综合性的计划时，必须能够将计划当作为单元整体加以表示和进行处理，框架和图式提供了这种较大单元的表示法。此外在归纳和改进各种行为规则中，必须记录和比较其他一些信息，图式表示法是一种合适的形式体系。

(8) 计算机程序和其他过程式编码：有些学习系统的目是获取某种能力，以便高效率地执行某一个特殊的过程，而不是对该过程内部结构的推理。大多数的自动程序设计系统都属这一类，除了计算机程序外，过程编码涉及人类运动的技巧，机器人操作器的指令序列以及其他机器的技巧。

(9) 分类学：根据观察的学习可以把全局结构化的领域对象归纳为分类体系，把对象描述分成群，组成新的分类，并形成层次分类要求系统建立有关的分类判据。

(10) 多重表示：有些知识获取系统对新获取的知识使用几种表示格式，例如有些发现和理论形式化系统，它们获取概念、

(4) 产生式规则：产生式规则是学习系统中广泛使用的知识表示形式，有4种基本运算可用来获取和改进产生式规则：创立（建立新规则）、一般化（削弱条件部分的限制，展宽规则的应用范围）、特殊化（增强条件部分的约束，限制规则只应用于个别场合）、组合化（把多条规则组合为一条较大的规则，以利于消除冗余条件或冗余的行动）。

(5) 基于形式化的逻辑表达式和相关的形式体系：这些通用的表示法已用于系统输入个体对象的形式化描述和系统输出生成概念的形式化描述。其形式是形式化的逻辑表达式，包括有命题、谓词、有限值变量、变量范围有限制的语句等。

(6) 图或网络：图和网络在许多领域中是一种较方便和有效的表示法，有几种学习技术利用了图匹配和图传递的模式来比较和标注知识。

(7) 框架和图式：当系统获取综合性的计划时，必须能够将计划当作为单元整体加以表示和进行处理，框架和图式提供了这种较大单元的表示法。此外在归纳和改进各种行为规则中，必须记录和比较其他一些信息，图式表示法是一种合适的形式体系。

(8) 计算机程序和其他过程式编码：有些学习系统的目是获取某种能力，以便高效率地执行某一个特殊的过程，而不是对该过程内部结构的推理。大多数的自动程序设计系统都属这一类，除了计算机程序外，过程编码涉及人类运动的技巧，机器人操作器的指令序列以及其他机器的技巧。

(9) 分类学：根据观察的学习可以把全局结构化的领域对象归纳为分类体系，把对象描述分成群，组成新的分类，并形成层次分类要求系统建立有关的分类判据。

(10) 多重表示：有些知识获取系统对新获取的知识使用几种表示格式，例如有些发现和理论形式化系统，它们获取概念、

(4) 产生式规则：产生式规则是学习系统中广泛使用的知识表示形式，有4种基本运算可用来获取和改进产生式规则：创立（建立新规则）、一般化（削弱条件部分的限制，展宽规则的应用范围）、特殊化（增强条件部分的约束，限制规则只应用于个别场合）、组合化（把多条规则组合为一条较大的规则，以利于消除冗余条件或冗余的行动）。

(5) 基于形式化的逻辑表达式和相关的形式体系：这些通用的表示法已用于系统输入个体对象的形式化描述和系统输出生成概念的形式化描述。其形式是形式化的逻辑表达式，包括有命题、谓词、有限值变量、变量范围有限制的语句等。

(6) 图或网络：图和网络在许多领域中是一种较方便和有效的表示法，有几种学习技术利用了图匹配和图传递的模式来比较和标注知识。

(7) 框架和图式：当系统获取综合性的计划时，必须能够将计划当作为单元整体加以表示和进行处理，框架和图式提供了这种较大单元的表示法。此外在归纳和改进各种行为规则中，必须记录和比较其他一些信息，图式表示法是一种合适的形式体系。

(8) 计算机程序和其他过程式编码：有些学习系统的目是获取某种能力，以便高效率地执行某一个特殊的过程，而不是对该过程内部结构的推理。大多数的自动程序设计系统都属这一类，除了计算机程序外，过程编码涉及人类运动的技巧，机器人操作器的指令序列以及其他机器的技巧。

(9) 分类学：根据观察的学习可以把全局结构化的领域对象归纳为分类体系，把对象描述分成群，组成新的分类，并形成层次分类要求系统建立有关的分类判据。

(10) 多重表示：有些知识获取系统对新获取的知识使用几种表示格式，例如有些发现和理论形式化系统，它们获取概念、